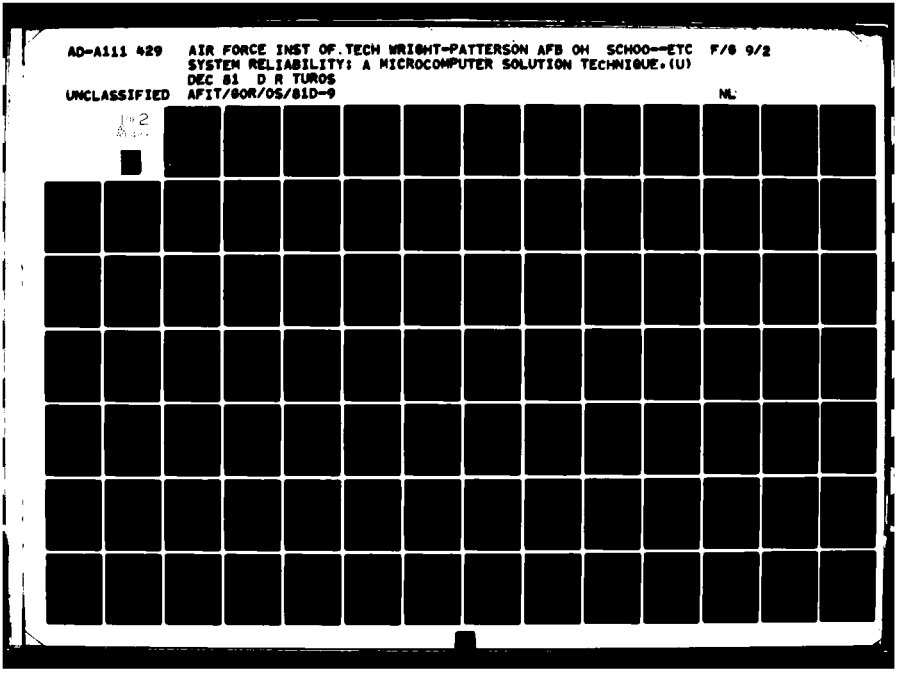
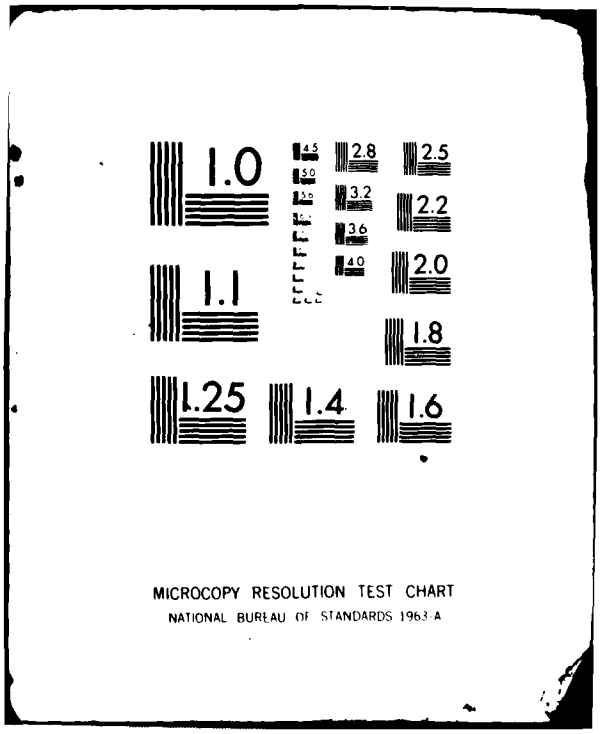

AD-A111 429
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
SYSTEM RELIABILITY: A MICROCOMPUTER SOLUTION TECHNIQUE.(U)
DEC 81 D R TUROS
UNCLASSIFIED AFIT/GOR/OS/81D-9
NL
1 of 2


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>AFIT/GOR/OS/81D-9 | 2. GOVT ACCESSION NO.<br>AD-A111429 | 3. RECIPIENT'S CATALOG NUMBER<br>429 |
| 4. TITLE (and Subtitle)<br>SYSTEM RELIABILITY:<br>A MICROCOMPUTER SOLUTION TECHNIQUE (1) | | 5. TYPE OF REPORT & PERIOD COVERED<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Air Force Institute of Technology (AFIT/EN)<br>Wright-Patterson AFB, Ohio, 45433 | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS | | 12. REPORT DATE<br>December 1981<br>13. NUMBER OF PAGES<br>142 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>Unclassified<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

DTIC ELECTE FEB 2 1982 H

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

28 JAN 1982

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE AFR 190-17.

Fredric C. Lynch

FREDRIC C. LYNCH

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer program
Reliability
Graphics
Pascal
Microcomputer

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This study investigates the feasibility of using a microcomputer to calculate system reliabilities. The computer program, written in Pascal on an Apple II computer, uses interactive graphics to allow a user to manipulate a system of components. The components are limited to a constant reliability in parallel and series structures. Once a calculation has been made, the user can then reconfigure the system and recalculate the reliability. This technique may provide insights to how the system reliability

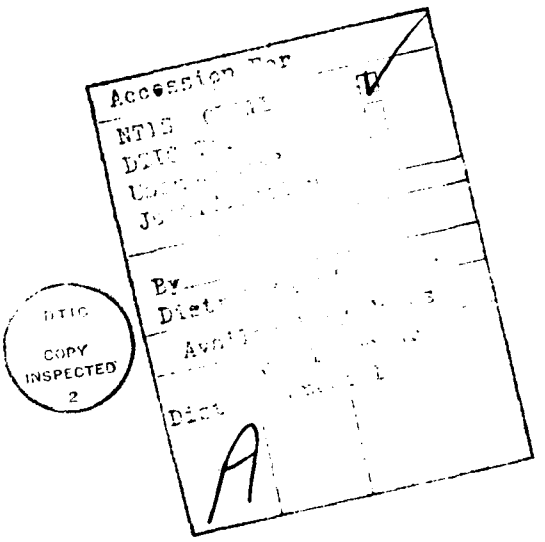
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

changes as components are changed.

The appendices include the system configuration, User's Guide, Programmer's Guide, and program listing.



AFIT/GOR/OS/81D-9

# SYSTEM RELIABILITY:

# A MICROCOMPUTER SOLUTION TECHNIQUE

THESIS

AFIT/GOR/OS/81D-9 Donald R Turos, Jr.
Captain USAF

AFIT/GOR/OS/81D-9

Thesis

# SYSTEM RELIABILITY:
# A MICROCOMPUTER SOLUTION TECHNIQUE

by

Donald R. Turos, Jr.
Captain USAF

Prepared in partial
fulfillment of the
requirements for a
Masters Degree

December 1981

School of Engineering
Air Force Institute of Technology
Wright-Patterson Air Force Base
Ohio

## Preface

I wish to thank my advisor, Major Daniel Fox, whose patience allowed me the flexibility to overcome my own procrastination and enable me to complete this effort.

I sincerely hope that being a member of a division championship team in both intramural softball and football is enough for the successful completion of my athletic scholarship.

Finally, only a good working relationship with a great group of GOR's and an understanding and supportive wife could have made my stay at AFIT so bearable.

## Abstract

This study investigates the feasibility of using a microcomputer to calculate system reliabilities. The computer program, written in Pascal on an Apple II computer, uses interactive graphics to allow a user to manipulate a system of components. The components are limited to a constant reliability in parallel and series structures. Once a calculation has been made, the user can then reconfigure the system and recalculate the reliability. This technique may provide insights to how the system reliability changes as components are changed.

The appendices include the system configuration, User's Guide, Programmer's Guide, and program listing.

# Table of Contents

## List of Figures

# SYSTEM RELIABILITY:
# A MICROCOMPUTER SOLUTION TECHNIQUE

## I Introduction

System reliability is a major question for a wide variety of people and concerns. It is easy to imagine a need for reliability calculations, without access to a machine that could provide the answer or where time to do the calculations by hand is not available. In the past, the large scale computer facility could provide such support for a specific, preprogrammed problems, but very often this support was not available to a small scale operation. Any needed computations would have to be carried out by hand: a tedious and error prone process at best. This study is an attempt to fill the gap by using an inexpensive microcomputer to provide an automated solution technique to the system reliability problem.

### Background

Topic Area. The main topic area, reliability theory, has received a reasonable amount of research attention. The area of interest important to this study lies with the work done with parallel and series structures of components. The mathematical equations used for determining the reliability of a system of components with known reliabilities are well documented in any number of

sources; Hillier and Lieberman (Ref 6) provide a reasonable discussion of these equations. Figure 1 presents the form of the reliability equations. This study will automate the evaluation of these equations to provide the reliability calculations for a user defined system of components.

---

$$\text{Parallel} = 1 - (1-R_1)(1-R_2) \ldots (1-R_n)$$

$$\text{Series} = R_1 \times R_2 \times \ldots \times R_n$$

where $R_i$ is the individual component reliability.

---

Figure 1. Reliability Equations

Related Research. Related areas of interest involve the choice of a microcomputer, graphic subsystem and language to support the study; then the application of the technology to provide a useful tool. Any of the many microcomputers on the market today could have been used to support this study. A decision was made to write the program in a standardized, high level programming language to allow the results to be used on as many different machines as possible. Machine availability was also a constraint, an Apple II was readily available. This particular microcomputer supports both Fortran and Pascal. The Apple machine was a good choice for two other reasons: the Apple II is widely availabile, and the UCSD Pascal

system supported by the Apple is becoming an industry standard (Ref 10:v).

The Pascal language was designed to aid the beginning programmer in learning programming techniques while affording the advanced programmer the power of an Algol like, block-structured language. Examples of block strctures are IF-THEN-ELSE, DO-WHILE, CASE-OF, REPEAT-UNTIL, and procedures. These structures of the language readily provide for self documenting code. Two important features of Pascal are the flexible data structures and recursion. Data flexibility allows the user to structre the data in the most convient form, depending on the desired objective; for example, minimum amount of memory per element, computation speed, access speed, or even readability. Recursion, calling a routine from within that routine, is very important aspect of reducing the code needed to handle problems, especially if the data can be structured to be handled by recursive routines. Two specific applications of both of these techniques will be discussed in Chapter 3. An excellent discussion of Pascal is provided by Jensen and Wirth (Ref 7). The Turtlegraphics subsystem of UCSD Pascal has been implemented for the Apple and supports all of the protocols of the UCSD Pascal System, thus providing the designer with a great deal of power and flexibility on the graphics screen. An excellent discussion of the Apple

implementation of the UCSD Pascal Operating System is provided by Lewis (Ref 10). The specific configuration of the Apple II system used for this study can be found in Appendix 1.

Several other areas were also explored before design of the program started. The first area was that of circuit theory (Ref 1, 5, 8). The computer has been used in several applications for generating circuits using graphics capability. Most applications are closed loop in nature and would not fit in with the network structure of the reliability system. The applications are most useful for circuit layout and positioning, but this would not be the case for this design.

The second area explored concerned algorithms designed to find the shortest path or K shortest paths through a network (Ref 9, 12, 16). Other algorithms dealt with finding the most reliable route and the N most vital links (Ref 13, 17). These articles most often were designed to aid the user performing manual calculations. The purpose of this program is to automate these calculations. Further search of the literature was abandoned as no specifics for the topic as applied to automated calcution using interactive graphics were found.

## Study Overview

Objectives. The objective of this study is to determine the feasibility of using a microcomputer for interactive reliability systems generation and calculation. Through making use of the graphics capability of the microcomputer, the user can view a representation of the system as it is generated. Success in meeting these objectives rely on the software providing the results of the user's input within a reasonable time . This time should not exceed a maximum of several seconds to preclude user dissatisfaction and abandonment of the system.

Scope-Limitations. The scope of this study will be limited to reliability systems with independent components and constant component reliabilities. The first assumption insures that each component reliability is not affected by any other component. This condition must be met to use the deterministic equations to solve for the system reliability. Because of this condition, the relaxation of this assumption is impossible for this study. The independence assumption is reasonable for a large portion of the reliability field. However, some of the work done in reliability theory concerns dependent components and is the way a significant number of real world systems behave. The second assumption is convenient for a feasibility study as many real world systems may be approximated in this fashion. This limitation may indeed be relaxed to allow

the reliability of any or all components to be a function of time. This is discussed in Chapter 3 as a possible extension to a more general solution of the system reliability problem.

Several other limitations exist. The intent of these limitations is not to limit the form of the system, but to provide reasonable limits for a feasibility study. In actual use, these limits could be easily changed to suit a specific user's requirements. An arbitrary limit of 100 components located on a 20 by 20 grid has been set for the study. Any single component may be in parallel with a maximum of six other structures. Any input series structure is limited to a maximum of eight components. The reliability of any individual component will be displayed as a two digit rounded percentage, but the program will allow an eight digit input reliability while the Apple system can maintain a maximum of 32 bits (approximately 7 significant digits) of accuracy for real numbers (Ref 3, pg 85). The program is not yet able to interface with a mass storage device, thus the saving of a generated system is not supported; this will also be discussed as a program extension. The program is interfaced with a printer to allow for a representation the generated system to be printed. This feature uses a standard 80 column printer format.

Hardware limitations also exist. The Apple allows for a matrix of 280 (horizontal) by 192 (vertical) addressable dots on the graphics screen. Thus the graphics capability only allows for a maximium of a 10 by 10 grid of components to be reasonably displayed at any one time. The Apple Pascal system has a limited amount of memory (44K 8 bit bytes) available for use by the programmer. This amount is reduced to 34K when using the graphics screen. The amount is not a critical limit, as the Pascal system allows the programmer to segment the code to fit the available space. This is at the cost of execution speed because the system must interact with the mass storage device to overlay the active code into memory. An intermediate version of the code did not require segmentation; but in the interest of follow on program modification, a decision was made to segment the program into four logical parts to insure memory was available for modifications and extensions. The time delays introduced by segmentation are not significant, as they occur at normal break points in the operation of the program.

## II Methodology

This chapter will provide an overview of the system reliability program. The topics to be discussed will be system capabilities, data structures, and rationale for segmentation. Finally, there will be a discussion of each main program segment, focusing on the procedures and the underlying algorithms. The discussion will be general in nature as the specifics of operation may be found in Appendix 2, the User's Guide, while the specifics of the program may be found in Appendix 3, the Programmer's Guide.

### Program Overview

Capabilities. The capabilities of the progam allow a user, with minimal knowledge of reliability, to generate a system of parallel and series structures, edit the system to user satisfaction, and calculate the reliability of the system. After each calculation the user has the option of obtaining a hard copy a representation of the system. The user may then edit the system to calculate the reliability of a slightly different configuration. This edit capability allows a user to conveniently perform online sensitivity analysis. The user may then reinitialize the system grid and construct a new system.

The program design allows for easy changes in capabilities. The limits, grid size and the number of components in the grid and each structure, are coded as Pascal constants. Thus, the data grid and data structures

can be modified by simply changing several constants and then recompiling the program. There are several procedures that currently contain no active code; this is to provide for future extensions. The desired code can be inserted into the procedure stubs because the existing structure already provides for the implementation of the extensions.

Data Structure. The data types available in Pascal are very flexible and can be adapted to meet almost any need. They are available in two different types, unstructured and structured. The unstructured types are constants and variables used in a program. These are very simple and are discussed in Appendix 3, the Programmer's Guide.

This program makes use of two of the structured types: the record and the array. The record structure, called a node in the program, was configured to hold the data for each component. Associated with each node is a pointer that is the starting location for the record in memory. Each node contains pointers to the next, the previous, and the parallel components to the component represented by the node. Each node also contains the component's integer grid (ordered pair) location, a two character identification tag, and a real number representing the component reliability. Each node can use up to fourteen words of memory depending on system constants. The current configuration uses twelve words per node. This memory is

dynamically allocated when a new node is inserted into the grid by the user. A maximium of 2500 words of dynamic storage is available to the user (Ref 4, pg 255), thus a realistic limit to the number of nodes is less than 200.

The array structure, called IJPOS in the program, provides the grid into which the components are inserted. The IJPOS array contains the pointer to the node that contains the data for the component at any (I,J) grid location. The grid is sized by two constants: one for the number of columns and the other for the number of rows. The program makes all checks for the grid boundaries against the constants, thus the user may configure the grid to any rectangular shape. The minimum suggested is a 10 by 10 grid because this is the maximum that can be displayed on the screen at one time. The maximum should be limited by the number of nodes needed in any direction, but the upper limit of about 200 active nodes must be kept in mind. The 20 by 20 grid currently employed works nicely for a variety of small scale systems and effectively demonstrates the capabilties of the program.

In practice, the grid system is really not limited to a maximum of 200 nodes. The user can always construct subsystems and calculate their reliabilities, then use each of those subsystems as a component in a macro-system. This type of application is limited only by the user's imagination and stamina.

Program Structure. The program has been divided into four sections, each operating under the control of the executive routine. Of the four sections, three are segment procedures and the fourth is an intrinsic unit. A segment procedure is an independent, self contained procedure that only refers to the procedure that called it. An intrinsic unit is a group of user written procedures that any program can use, and is located in the System Library file. The System Library file contains special purpose system provided functions and procedures such as trigonometric functions, math functions, graphic routines, and any other special purpose type routines in addition to any user written routines. The three segment procedures and the executive routine compromise the main program and are compiled into a single code file. The intrinsic unit, named Basics, is compiled separately and then inserted into the System Library file. This strategy was taken for two reasons: first, the Apple only has a limited amount of space to build the text file that will be later compiled into a code file; second, the Apple only has a limited amount of memory for the loading and execution of a code file.

The first reason comes from the fact that the Apple Pascal editor has room for only 18400 bytes of information (Ref 4, pg 98). This limits a programmer to between 600 and 800 lines of text in the editor at any time. The

program contains about 1600 lines of text, excluding comments, thus there was a need to break the program into pieces. During prelimenary coding, this problem was solved using a technique that split part of the code into the intrinsic unit. This division split the program into two main sections and reduced the compilation time of the main program, because an intrinsic unit is compiled separately. The limitation on text file size can also be overcome by using the compiler INCLUDE option. This option allows the programmer to signal the compiler to go to another text file and compile that code and then return to the current text file and finish compiling the rest of the program. This option was used to break the main program's 1100 line text file into three pieces: one containing the executive routine and the edit segment, one containing the initialize segment, and one containing the analyze segment. This technique was not needed for the Basics unit, since that text file can fit into the editor.

The second reason, limited memory for execution, was aslo solved with the solution to the limited editor capability. The main program's three sections were made into segment procedures. This option of the Pascal system overlays the segment's code into memory only when a routine from that segment is active. The optimal choice for segmentation dictates that the segments be independent and infrequently exchanged. The choice of segments abided by

that rule. The initialize segment is only used to set up the grid and the display, and then is discarded. The edit segment performs both the system generation and edit functions and is the largest segment. The analyze segment performs the reliability calculations and outputs the system to a printer. The edit segment and the analyze segment are only exchanged when the user wants to calculate a reliability. The time spent overlaying either procedure is minimal and is completed as the next prompt is displayed on the screen. The time delay is short because a substantial amount of the routines needed by the edit segment are in the Basics unit; this makes the size of the edit segment smaller, thus making the time needed to overlay that segment into memory shorter.

A final reason, documentation, while not critical to the program operation, is an important consideration for the programmer and user of the system. The top-down structuring allows for the above type modifications without loss of continuity in the program. This structure also allows for searching only a portion of the code for an error. Other benefits include leaving procedure stubs to be completed later and easy reading and locating of any portion of code.

Segments

This section will discuss the specific procedures and structure of the Basics unit, the initialization segment,

the edit segment, the analyze segment, and finally the executive routine.

Basics Unit. The Basics unit contains four important subparts: the global variables, general use procedures, display procedures, and cursor component movement procedures. The global variables include the system constants, the two structured types, and the system variables. All of these items are available to all procedures; each procedure may also declare variables that are only active within that procedure. The global variables serve as the communications links between the major segments and the Basics unit.

The second part of the Basics unit is the general purpose routines. These routines are used by at least two different major segments, thus are placed in this common area to avoid duplication. The routines include all of the procedures used to interact with the user on the console and the procedure needed to define a new node. None of the procedures are unique and many could be used in any program that needs to interact with a user via the text screen or graphics screen of the Apple. Several of the routines were borrowed from the sample programs provided with the Apple Pascal System. Those routines are specifically mentioned in the computer code and the Programmer's Guide.

The third part of the Basics unit contains the display procedures used specifically by the edit segment. These

procedures are designed to operate under any of the possible configurations of the system constants. Two of the functions are used to determine the X and Y location of a specific component on the graphic screen window. One routine determines the 10 by 10 subgrid that will be displayed at any one time. This routine is invoked if the component cursor moves out of the currently displayed subgrid. The two most important routines are used to display an individual component in either white on black or black on white and the entire system. The system display routine uses the component display routine to place each component into the window. The routine then draws the line to the highest parallel component in the window (this component has not yet been drawn), and then draws the line over and down (if necessary) to the next component. This procedure is repeated until all components in the current window are displayed. The procedure is quite fast, a typical screen is displayed in 3 to 5 seconds. The delay provides the user some time to plan his next addition.

The final part of the Basics unit contains the two procedures used to move the cursor up and down, and left and right on the screen. These procedures are also used exclusively by the edit segment.

The Basics unit is a special case because it is a complete program except for an executive routine. The main program serves as the executive. As such, the unit needs

to be recompiled only if there is a change to any routines or constants. The code is then linked into the system library using the system librarian package. The main program does not have to recompiled if there is a change to the Basics unit and likewise the Basics unit does not have to be recompiled when the main program is changed. The main program can be nonexistant when the Basics unit is compiled because the two parts are linked only when the main program is compiled or executed. This feature allows the programmer to deal with only half the code at any one time; if the procedures in the Basics unit are fully operational the unit will remain static while experimentation and change occurs in the main program. Specifics on using the system librarian package to insert the unit into the system library can be found in Appendix 3, the Programmer's Guide.

Initialize Segment. The initialize segment contains the routines needed to prepare the program for operation and allow several default parameters to be changed. This segment is also executed if the user wants to clear the system grid and start working on a new problem. This segment is also responsible for preparing the instructions on the text screen that the user can display if help is needed on a program command. This segment also contains the procedure stub that will eventually interface with a mass storage device to provide for reading an old

reliability system back into memory. The segment allows the user to change the default reliability used in defining new nodes and also allows the user to turn the automatic labeling option on or off.

Edit Segment. The edit segment contains the procedures needed to generate and edit the reliability system. The generate section includes the procedures to generate a parallel and series structures and to label a component. The edit section includes the procedures to remove or change a component and control the cursor, which is the current component. The executive part of this segment selects the generation sequence, if data does not exist in the grid, or the edit sequence, if the grid contains data.

The generation sequence uses recursive calls to the parallel and series procedures. The user begins the sequence by choosing the original structure of the system, either parallel or series. The program then displays that structure and locates the cursor component at the first component in the structure. The user is then asked to either label the component or change the component into the other structure. The program inserts the new structure or labels the component and continues to prompt until the structure is finished. The program then returns to the second component of the original structure and repeats the process until the entire system is completed. The program

then enters the edit sequence.

Once in the edit sequence, components can be changed, removed or added. The quit option allows the user to exit to the main program executive routine which then allows a transfer to any of the three segments. The user can toggle the labeling option and change the default reliability. The generation sequence can be entered by using the change option. The procedures in this segment are relatively straight forward and easy to follow once the notion of recursion is understood. Simply stated recursion is the act of calling a procedure from within itself or calling a procedure that called the procedure now executing. The Apple Pascal system fully supports this feature of the UCSD Pascal system, and allows the programmer a great deal of power from a relatively few number of procedures while providing for the linking of all the recursive steps automatically.

Analyze Segment. The final segment is the analyze segment, which controls the reliability calculation and provides the interface to the output devices. This segment also uses recursion to implement the mathematical equations to calculate the reliability of the system. This sequence starts at the first component and calls the multiply procedure. If the component is in parallel with any structures, the procedure to analyze a parallel structure is invoked. This procedure must recursively call the

procedure to analyze a series structure because a parallel structure is made up of series components. The series procedure recursively calls the multiply procedure to calculate the series reliabilities. The multiply procedure combines the results of the parallel structure with the original component and then procedes to the next series structure. Recursion allows three relatively simply procedures to be able to reduce any system no matter how complex. This ease is also aided by the structure of the data and the grid it resides in. The speed at which the reliability of a moderate sized system (50 components) can be calculated is no more than a few seconds.

The second function of the analyze segment is the interaction with external devices. The primary device is a printer. The grid will be printed out in 10 by 10 blocks in a standard 80 column format. This routine can easily be modified for 132 column output. The procedure to store a reliability system to disk is not provided but a procedure stub exists for this purpose.

Executive Routine. The executve routine is very simple as it must provide only for the switching between the various procedure segments, which all have their own executive routines which function in the same manner. This executive sequence is basically the same for all segments and is as follows:

1. The display is updated.
2. The menu is presented.
3. A command is selected.
4. The command is executed.
5. Repeat until the exit command is selected.

The executive routine exists in the main program text file along with the edit segment. When this file is compiled, the correct version of the Basics unit must exist in the system library or the compilation will fail. It is at compilation time that the initialize and analyze segment text files are included and compiled with the main program text file into the single main program code file.

# III Results

This chapter will cover the validation of the computerization of the reliability equations, the conclusions drawn about the system's feasibility, the major problems encountered, and several extensions to enhance the system.

## Validation

The validation of the computerized reliability equations is an important part of the study. The method of validation chosen was to enumerate the possible combinations and configurations of components on a small scale and insure that the computed reliability equaled the hand calculated reliability. This strategy was selected for two reasons: first, recursion insures that the same sequence is used to compute the reliability of a configuration no matter where it is in the system; and second, hand calculation, as has been mentioned, is tedious and error prone procedure. A constant reliability of 0.5 made the hand calculations simpler, but effectively tested the sequence of computations the program must carry out. Simple series and parallel structures were tested first; after verifing the computations, more complicated structures were tested. Thus the hand calculations were kept to a reasonable limit. The configurations used are depicted in Figure 2.

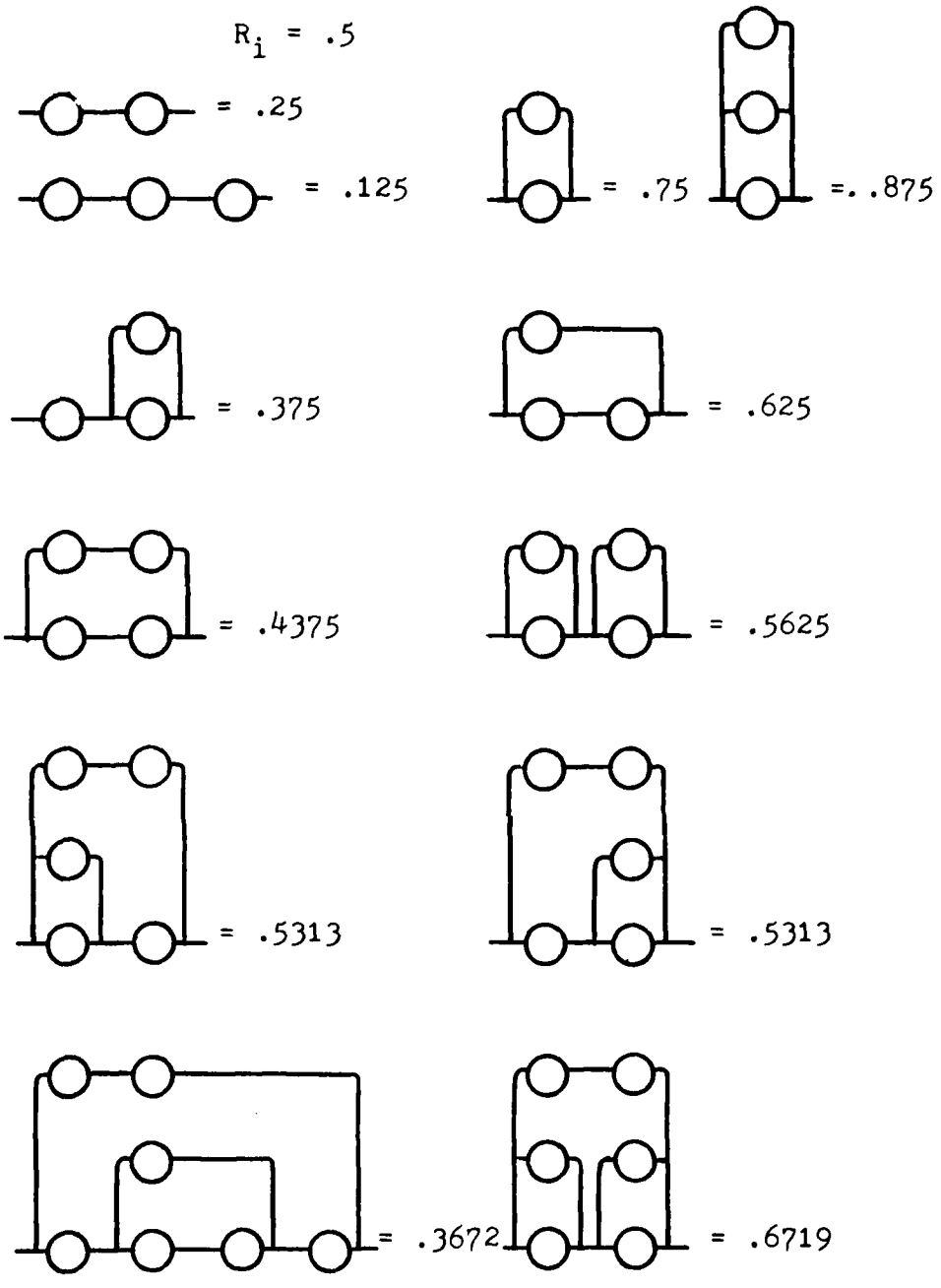



**Figure 2. Component Configurations for Validation.**

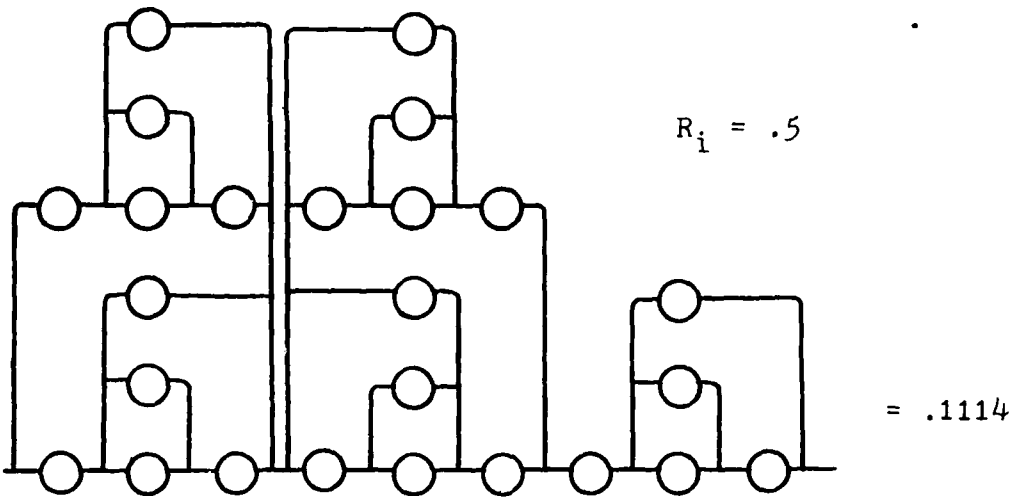



Figure 3. Prototype Configuration for Validation.

A single large system was configured and a reliability was calculated. The computed number agreed with the number calculated very carefully by hand. This prototype configuration is presented in Figure 3. The calculation changed appropriately when the component reliabilities were adjusted. The program was also tested using the possible extreme values of 0.0 and 1.0 resulting in no errors. The strength of the tests performed indicate that the program does perform the reliability calculations correctly.

## Conclusions

Several conclusions can be drawn from this study. First, and foremost, this program shows that a microcomputer can indeed be used to interactively generate reliability systems and then calculate the reliability of the system. The use of the graphics capability allows the user the ability to visualize and then build and change the system interactively on the console. The actual implementation is not important; it is the power and flexibilty that an interactive system provides that was the underlying issue of this study.

Second, the microcomputer is a powerful and useful tool -- useful in a wide range of applications. The study has shown that a microcomputer can handle a complex problem with relative ease. The available graphics allow a very user oriented display of the reliability system.

Third, the UCSD Pascal System as implemented on the Apple II System is a very powerful and useful tool for a programmer and user of microcomputer systems. This system offers most of the scientific computing power of Fortran less exponentiation with the block-structuring of Algol and the self-documenting feature of Cobol. The operating system allows the programmer and user a great many options and flexibility. The power of the segmenting and overlaying capabilities are shown because the program, which taken as a single piece could not fit into the

available memory, can execute with a surplus of space.

Finally, the most important conclusion is that microcomputer systems can be used to handle complex problems in a highly desirable fashion -- interactively. The power of this avenue of approach to any problem can not be overemphasized. A well written program can be used as a powerful aide to a manager or technician that does not have the time to manually tackle a problem.

Problems

Several problems were encountered during the course of the study. A class of problems was computer implementation dependent and as such, the discussion can be found in Appendix 3. The first major problem to be encountered was the limited size of the system editor. This problem was solved by using the compiler INCLUDE option and then dividing the program into several parts with no harmful effects. Editing four different pieces of code can be cumbersome but the Pascal Operating System allows for smooth transitions and storage. The floppy diskettes can be unreliable, a single instance of data retieval error occured in several months of operation. The operating system allows for duplicating diskettes before an unrecoverable error can occur. The user must make at least one backup copy to prevent the catastrophic loss of a complete diskette, as disk units have been know to fail.

A second problem concerned the graphics capability of the Apple System. The design of the individual component and the hardware combined to limit the display of only a 10 by 10 grid of components. This is only a quarter of the possible grid. The solution rested on software to automatically keep only the portion of the grid that the user is working with on the screen. This strategy causes the screen to shift radically if the grid is sparse because the cursor component shifts large distances as the user moves through the grid. The addition of a position prompt of the current component, that is on the screen at all times, relieves most of the confusion that can result from the computer's attempt to display the grid.

A final problem, that was really a limitation of the system, was the amount of time needed to compile the program. The compile process takes about five minutes for the Basics unit and about eight minutes for the main program. This was an annoyance when working but was probably a small price to pay. This was the only drawback to the Pascal operating system; the processor can only compile 150 lines per minute (Ref 3, pg 68), the more lines the longer it takes. This inconvience must be put into perspective: batch turn around for a large scale computer is normally several minutes but can take days depending on the work load and the environment. Even in an interactive environment, the user may spend an inordinate amount of

time waiting for a listing at a production control window. The longest the microcomputer will ever take is a few minutes. This inconvenience happened only during development as the program was constantly being recompiled, the user will normally execute the compiled code unless a change must be made.

Extensions

Two major extensions will be discussed in this section, several other minor changes are mentioned in Appendix 3. The first extension is the addition of the routines to store and retrieve data to a mass storage device. This was not accomplished because the original program design precluded having a node for every grid location in memory at the same time. This design allowed flexibility in dealing with the limited amount of memory. The program currently executes with a surplus of memory, thus the extensions should be attempted.

The strategy could be one of combining the IJPOS grid array and the node records into a single structure - a two dimensional array of record. The pointer can then be changed to a (I,J) address in the grid. Once the new structure has data in it, input or output to mass storage is very easy: read or write the individual records. The difficulty lies in sizing the static array (about 12 words per node) to fit into the available memory, approximately 6000 words minus the amount needed for the largest code

segment. This sizing can force only a limited size system at a time, but the storage capability coupled with the decomposition of the reliability system may counterbalance the negative effect. This option requires the revision of every reference using a pointer to using an ordered pair address; this will be a very time consuming and tedious effort, but the results may very well be worth the effort. This method implies a static array of nodes but another method is available.

In this second option, the record structure could add a single pointer to be the location in memory of the node. The pointer would only be used to allocate or deallocate the memory needed for the node. This option would preserve the efficient use of the dynamic memory available, yet still allow for a simple interface to the mass storage device.

The second extension would allow the individual component reliabilities to be a function of time. This could be implemented by providing the program with several probability distribution functions, such as the Wiebel or Exponential that could be used to model the reliability. The identification label could be used to indicate that the relibility is a function of time and the reliability could be used as the function's parameter. An alternate solution would use the ability to declare variable record formats; thus one type for a constant reliability and another type

for the different functions allowed, since some are characterized by more than one parameter. The analyze segment could be modified to to ask for the duration and interval desired for the resulting system reliability; then the analysis routines could call the appropriate function (one could be a constant) for the reliability using the current time and parameter set for each component in the normal sequence. The reliability calculations could be saved for each interval and then plotted on a graph of the system reliability versus time using each interval calculation as a data point. The number of functions will probably not be limited by memory size because the analyze segment is smaller than the edit segment. This extension can remove a restriction that forces an assumption to be made that limits the analysis that can be conducted on many real world problems being investigated using this program.

## Bibliography


1. Albrecht. "Evaluating Systems Reliability", IEEE Spectrum,15:43-7 (Aug 1978).

2. Apple II Reference Manual. Apple Computer Inc., 1979.

3. Apple Pascal Language Reference Manual. Apple Computer Inc., 1980.

4. Apple Pascal Operating System Reference Manual. Apple Computer Inc., 1980.

5. Bell. "Computer Aids For Reliability Prediction And Spares Provisioning",Electronic Communication,54: 136-42 (1979).

6. Hillier, Fredrick S., and Gerald J. Lieberman. "Chapter 14: Reliability", Introduction to Operations Research, 3rd ed. San Francisco: Holden-Day Inc., 1980.

7. Jensen, Kathleen, and Niklaus Wirth. PASCAL User Manual and Report, 2nd ed. New York: Springer-Verlag, 1975.

8. Klingman. "NETGEN, For Generating Large-Scale Network Problems", Management Science,20:814-21.

9. Lawler. "Computing The K Best Solutions", Management Science,18:401-5.

10. Lewis, Theodore Gyle. Pascal Programming For The Apple. Reston, VA: Reston Publishing Co. Inc., 1981.

11. Ostroski. "Run System Studies on a Microcomputer", Electronics World,190:56-7 (Oct 1, 1978).

12. Pollack. "The Kth Best Route", Operations Research,9: 578-80.

13. Ratliff. "Finding The N Most Vital Links", Management Science,21:531-9.

14. Rondeau. "Short-Cut Reliability Analysis", Machine Design,50:108-12 (Sep 21, 1978).

15. Shogan. "Reliability of a Stochastic Network", Operations Research, 24:1027-44.

16. Shorack. "Most Reliable Route Algorithm", Operations Research, 12:632-3.

17. Yen. "Finding The K Shortest Loopless Paths", Management Science, 17:712-6.

# Appendix_1

## SYSTEM_CONFIGURATION

# System Configuration

## Study Configuration

1. Apple II Plus with 48K memory
2. RAM card with 16k memory (required) (slot 0)
3. Apple Disk II (2) with controller card (slot 6)
4. Printer with controller card (slot 1)
5. Color TV with RF modulator

## Alternatives

1. Apple II with 48K memory (48k minimum)
2. Any number of disk drives (1 minimum)
3. Any printer with controller (not required)
4. Any 80 column or lower case card (slot 3)
5. Any video monitor

# Appendix 2

# USER'S GUIDE

## Overview

This User's guide is a discussion of the System Reliability program. This guide covers the general system, the set up process, the initializing process, the editing process, and the analyzing process. A quick reference command list, segment interaction chart, and example session are also provided. This guide should be used in conjunction with the Apple Pascal Operating System Manual and the System Reliability Programmer's Guide.

## Table of Contents

# User's Guide

This User's Guide will take the form of a structured walk through the workings of the program. The guide will be operational in nature and focus on the options the user may employ along with an explanation of the menu choices. The explanation of the inner workings of the program can be found in Appendix 3, the Programmer's Guide. The User's guide will address the following topics: general information, start up, initializing, editing, analyzing, and the main menu. This guide assumes a rudimentary knowledge of the Apple Pascal Operating System; a familiarity with the System and FILER commands is necessary. The guide can be read through as the user practices with the program or beforehand as a program familiarization step. The quick reference command chart provides a list of all available commands. The segment interaction chart provides a map of the paths available to the user. The example session is a tutorial on the program functions and commands.

## General Information

The program will allow the user to place up to 100 components on a 20 by 20 grid. The individual component will be represented by a 2 character identification label above a 2 digit rounded reliability percentage. This

component will be connected to other components using straight lines. The display will be constructed from left to right and bottom to top. The screen will either contain text information or a display of a 10 by 10 portion of the reliability grid. The bottom display line will be used to communicate information to and elicit responses from the user.

The program will ring the buzzer if an error has occured. A single sound indicates that the input character is not allowed from that menu. The corrective action is to reread the menu or use the <ESC> key to display the available commands. Two buzzes indicate that the limits of the grid have been exceeded. This means that a particular column or row has grown to large and no more components can fit, or an attempt has been made to put more than six components in parallel. In either case, the program will automatically insert as many of the components as is possible, the excess will be ignored. If no room exists then the program will offer the only option that can be successful completed. If the the maximum number of components is exceeded the program will remind the user each time an attempt is made to add a new component, but will not terminate the program. The limit is currently set at 100 but there is memory available for at least 200 components. This value can be changed; see the Programmer's Guide for specific details.

## Start Up

This program requires an Apple II with 48K of memory and a language or expansion RAM (Random Access Memory) card installed in slot zero, a single disk drive with controller card installed in slot six, and an appropriate monitor or television and modulator. A printer with interface card installed in slot one and a second disk drive attached to the controller card in slot six may also be used. If the Apple has the shift key modification, then select the normal keyboard (shift m generates a < ] > right square bracket). The user must insure that the correct library exists on the boot diskette. The SYSTEM.LIBRARY file must contain 45 sectors, if it does not or you suspect the file is not correct, please refer to the Programmer's Guide for specific instructions. Depending on the number of disk drives available, the start up procedure varies slightly. Once either procedure has been accomplished, the user can then EXECUTE the SYSREL.CODE file, and the initializing process will begin.

Single Drive. The single drive user must insure that the SYSREL.CODE file is on the boot diskette. If it is not, the FILER must be used to transfer the file from the WORK: diskette to the APPLE1: diskette (boot diskette).

Dual Drive. The dual drive user must only insure that the APPLE1: diskette (boot diskette) is in drive 1 and that the WORK: diskette is in drive 2.

Initializing

The initializing process is the first segment of the program to be encountered, and as such it informs the user about some general program limitations and functions, and enables the user to change program default values. An important piece of information is that the current component the program is working on is displayed as white letters on a black background; any other components are displayed as black letters on a white background. The current component will also be refered to as the cursor component.

Two important key stroke commands are also mentioned: the <ESC> (escape) key will display a quick reference command list, and the < <- > (left arrow or backspace) key will let the user change a selected command. The left arrow key allows the user to reenter a mistaken command choice for multiple keystroke entries (examples are P#, S#); the function will not work if the command is only a single key stroke (examples are E, C), other means are available to stop the commanded action.

After reading the information the user can then select the <RETURN> key to change the default values or any other key to continue the initializing process. Changing the defaults simply involves answering the questions. The autolabelling option is normally turned on and the default component reliability is 0.5. The user is now asked if

data resides on disk; this function is not yet implemented so either answer will cause the program to finish the initializing process.

If the user reenters the initializing process from the main menu, pressing the <ESC> key returns the user to the main menu without clearing the current reliability grid. Any other choice will erase whatever system had been constructed by causing the program to reinitialize. The rest of the process will be repeated as before, then the user automatically enters the generate sequence of the editing process.

Editing

The editing process consists of two sequences: generate and edit. The generate sequence is entered automatically when the initializing process is complete. At the completion of a generate sequence, the edit sequence is initiated. This entire process may also be initiated from the main menu. A generate sequence may be started from the edit sequence. The # symbol is used to indicate a single digit input is required to complete the command. The numbers in the lower left corner of the display indicate the row and column of the current component. This information is displayed at all times and is designed as an aide in keeping tract of the user's location in the grid.

Generate. The generate sequence allows the user to create a reliability system. The generate menu allows the user to insert a P)arallel structure with # components, a S)eries structure with # components, L)abel a component with an identification label and reliability, or move to the N)ext component in the structure. The current component limit for a parallel structure is six, while the current limit for a series structure is eight. These limits may be changed; see the Programmer's Guide for specific procedures. This process of inserting and adding components continues until the user has visited each component at least once. If the component does not have to be changed into a parallel or series structure, or the user is satisfied with the label, the N)ext command may be used to advance the cursor component. The program automatically advances to the next component after any other command.

It is important to note that the user's initial command choice, parallel or series, will determine the structure of the final system. This choice has a significant impact on the reliability of the system. If the user chooses the parallel command then the system will consist of # structures in parallel with each other. If the user chooses the series command then the system will consist of # structures in series with each other. This basic arrangement is unalterable; the system would have to be reinitialized to change the arrangement. In either

case, the structures may be very complex, but will still have the basic pattern. Based on this information, the suggestion is offered that the user choose a series structure as the first command until the capabilities of the program are understood. Any excess components can be later removed from the system by the edit sequence. A series component can also be effectively eliminated from the structure by setting the reliability to 1.0, while a parallel component can be effectively eliminated from the structure by setting the reliability to 0.0. At the completion of the generate sequence, the edit sequence is automatically initiated.

Edit. The edit sequence is entered automatically from a generate sequence or can be entered from the main menu. This sequence enables the user to change the system that has been placed on the grid. The edit menu allows the user to R)emove a component, C)hange a component or the defaults, move the cursor, or Q)uit and return to the main menu to procede to the analyze segment.

The R)emove command will attempt to remove a component from the grid; the routine is not very sophisticated and therefore can not handle all possible cases. The details of the command restrictions are discussed in the Programmer's Guide in the Internal Section: Main Program, REMOVE procedure.

The C)hange command displays the change menu which allows the user to start a generate sequence by using the P)arallel or S)eries command, L)abel a component, or change the defaults by using the T)oggle command to turn the autolabelling option on or off, and the D)efault command to change the component reliability default. The Q)uit command returns the user to the edit menu after the use of the T)oggle or D)efault commands, or if the C)hange command was inadvertantly selected. The program automatically returns to the edit menu after the completion of the P)arallel, S)eries or L)abel commands.

The cursor movement commands allow the user to move from component to component to make editing changes. The command keys were selected because they closely resemble a diamond shape on the keyboard. The positions in the diamond represent the direction the cursor will move: I and M force the cursor up and down respectively, while J and K force the cursor left and right respectively. The direction is followed by the number of grid locations to shift. The program will allow movement out of the currently displayed subgrid and then present a new section of the grid with the target component centered. The maximum input allowed is a 9. If no component exists at the target location, the program checks either side for a component; if a target still does not exist, the program reduces the shift amount and tries again until a target

component is found. The H)ome command will position the cursor at the lower left corner of the grid with the corresponding quadrant of the grid di olayed.

The Q)uit command will return the user to the main menu. This command can be used with the cursor in any position. The <ESC> key will display a quick reference list of all of the available commands. The left arrow < <- > key functions to change a command that requires more than a single character entry. After exiting the editing process, the user can then select the main menu A)nalyze command in order to calculate the system reliability.

Analyzing. All of the analyzing process options are initiated from the main menu. The A)nalyze command will cause the program to calculate the reliability of the system currently on the grid. A short pause can be expected while the calculations are progressing. The answer will be displayed and the <RETURN> key will call the print routine while any other key will return the user to the main menu. The P)rint command will begin the process of printing a formatted representation of the grid on a piece of paper. The routine places each component in its grid location with the row and column of the component, the row and column of the forward linked component, and the row and column of the backward linked component; the row of the parallel linked components; and the component reliability. The user must insure the printer is on and ready to accept

data. The analyze routine may be called from the print routine. The S)tore command will initiate the process of saving the grid on a diskette file but is not currently implemented.

Main Menu. The main menu commands allow the user to select which process will be initiated next. The commands are for the E)dit process, the A)nalyze process, or the I)nitialize process. The P)rint and S)tore commmands are part of the analyzing process but are selected from the main menu. As suggested by the command names, each selection places the user under the control of the specified process. The Q)uit command will cause the program to terminate and place the user in the Pascal Operating System command line.

The program is designed to allow the user to alternate between the E)dit process and the A)nalyze process at will. This facility allows the user to rapidly reconfigure a reliability system, calculate its reliability and then iterate the process as many times as is required.

## Quick Reference Command List

### Main Menu Commands

E)dit Process

#### Editing Commands

Generate Sequence
- P)arallel Structure
- S)eries Structure
- L)abel Component
- N)ext Component

Edit Sequence
- R)emove Component
- C)hange Component
  - P)arallel Structure
  - S)eries Structure
  - L)abel Component
  - T)oggle autolabelling on or off
  - D)efault Reliability
  - Q)uit Change
- I or M - cursor up or down
- J or K - cursor left or right
- H)ome
- Q)uit Edit Process

A)nalyze grid reliability

Ability to print grid

P)rint grid to paper

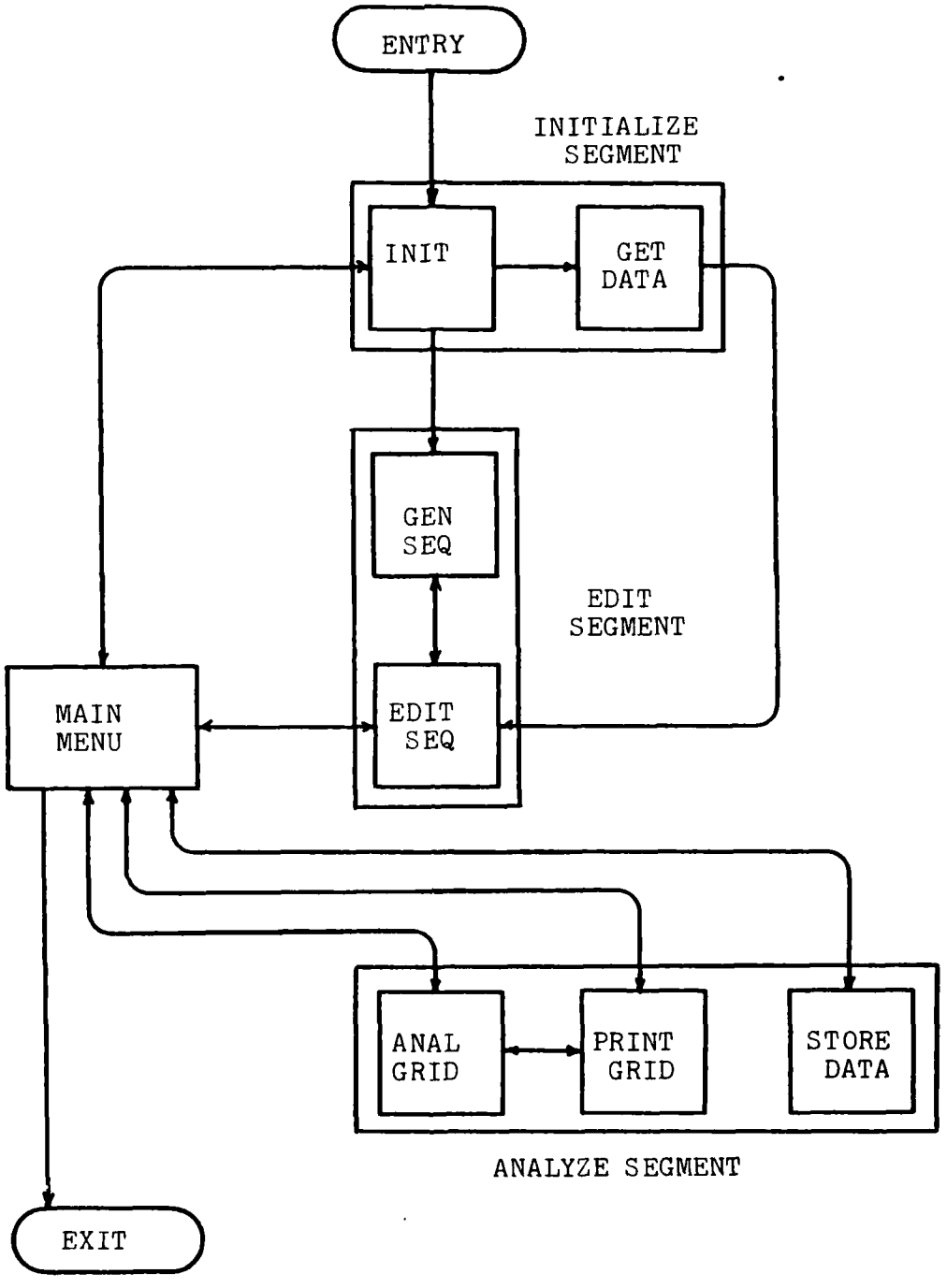
Ability to analyze grid

S)tore grid to disk

I)nitialize Process

Q)uit program

## Segment Interaction Chart



## Example Session

This example session contains a very simple system to present some of the commands of the program in a structured environment. The user must experiment with the prpgram to make it do the things that are of special interest. The example will recreate the system depicted in Figure 1.

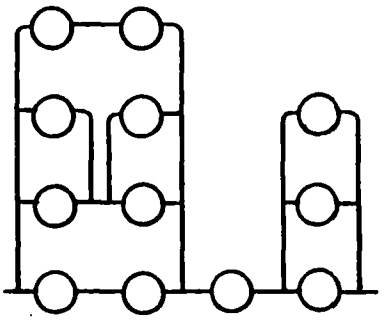



Figure 1. Example System

The example will be discussed exactly as the program will execute, thus the user should be doing the commands on the computer as this example is being read.

1. The first step is to insert the APPLE1: diskette into drive 1 and the WORK: diskette into drive 2. The Apple should then be turned on and the drives will operate, finally displaying the UCSD Operating System command line. In response to this the user should now EXECUTE the

WORK:SYSREL file and the system reliability program will begin to execute.

2. The user will first see the specific program information. Take time to read all of the comments as they are important to the correct operation of the program. After reading the comments, hit any key to continue with the program. The program now asks if the user has data stored on disk. Respond no to this question as the routine has not yet been implemented. The drive will again operate as the program starts the initializing process and display an appropriate message. At the completion of the initialization the program will display the generate sequence menu.

3. The user can now start to build the system depicted in Figure 1. The first step is to notice that the system is basically three structures in series. The first response is S3. The program now displays a three component series structure with the first component as the current component (white on black). Now, the first structure of the series is basically three structures in parallel, thus the second response is P3. The lowest component of the parallel structure is two components in series, so the next command is S2. The lowest structure is now complete so two N commands are used to move the cursor to the second element of the parallel structure. This structure is basically two structures in series so the next command is

S2. Now use a P2 command and two N commands, twice, to change the two series components into parallel structures and move the cursor to the top parallel structure. The last stucture of the parallel structure is a series structure so use a S2 command to finish the last structure and two N commands to move to the second component of the original series structure. Use another N command to move on to the last component in the series as the second is not to be changed. The last component is a simple parallel structure, so use a P3 command to finish the system. The generate sequence must now be exited so use three N commands to start the edit sequence.

4. The edit menu is now displayed, but the system is complete so use the QUIT command to exit to the main menu. Once in the main menu the ANALYZE command can now be used to calculate the system reliability. This process takes a few seconds and the program displays an appropriate message. The calculations are completed and displayed to the screen, reliability is .3298. Not being finished, use the space bar to return to the main menu. Once in the main main, the user can now return to the EDIT sequence to change the system.

5. Once back in the edit sequence, the user can now use a K2 command to move to the lone series component to change its reliability. Use the CHANGE command to enter that menu and then select the LABEL command and change the

component's reliability to .9, which the display automatically updates. Now use a K1 and I3 commands to move to the third component in the simple parallel structure. Use the REMOVE command to delete this component from the system. Again the display is automatically updated. Use the QUIT command to return to the main menu to analyze the new configuration.

6. Use the ANALYZE command to again calculate the reliability, which is now computed at .5089. This number should make intuitive sense as the .5 reliability was changed to a .9 reliability but a triple parallel structure was reduced to a double parallel structure somewhat negating the effect of the reliability change. Again use the space bar to return back to the main menu and then enter the edit sequence.

7. Use a K2 command to move over to the single series component. Use the CHANGE command and the select the DEFAULT command and change the default reliability to .9. Now, use a P4 command to change the single component into a simple quadruple parallel structure. Four N commands must be used to return to the edit menu. Now select the QUIT command to again return to the main menu to analyze the reliability. The reliability is now ..5654 and this is a final form of the system. So, insure the printer is turned on and use the <RETURN> key to print a copy of the system to paper. If no printer is connected to your system just

return to the main menu. Once the printer is finished use the QUIT command to exit the program. The final version of the system is depicted in Figure 2.

Note. This is a very small example, the user must feel free to experiment and use all of the commands to become familiar with them. The program offers numerous ways to cancel a command if a wrong choice was selected. No example can depict all of the facets of a system, only usage can teach the full range of capabilities.

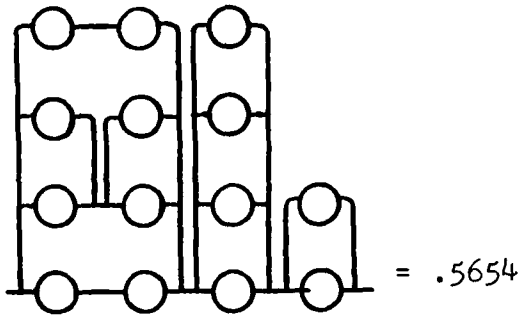



Figure 2. Final System

# Appendix 3

# PROGRAMMER'S GUIDE

## Overview

This Programmer's Guide will provide general information on the program and diskettes. Specific detail of the external operation of the program will include the editing, compiling and linking of the program segments. Specific detail of the internal workings of the program will include the Basics unit and all program segments. This guide should be used with a Pascal reference and the Apple Pascal Language Reference Manual.

This guide contains Appendix A, the program listing.

# Table of Contents

## Table of Contents

## Programmer's Guide

The Programmer's Guide is for use if the program must be changed or recompiled, or if the user desires an indepth knowledge of the internal program structure. This guide will begin with some general information and comments about the program. The rest of the guide is divided into three sections: external, internal, and program listing. The external section will discuss the external operation of the program. The topics will include editing and compiling of the main program; and editing, compiling, and linking of the Basics unit. The internal section will discuss internal workings of the program. The topics will include the Basics unit and each segment of the main program, executive, initialize, edit, and analyze. The program listing section will include a procedure hierarchy chart along with an annotated listing of the program.

Program Detail. This program has been divided into two major portions: the main program and the Basics unit. This division occured because the Pascal editor can only handle about a maximum of 800 lines of code. The Basics unit is compiled separately and is then linked into the system library file. A change to the unit does not require a recompilation of the main program. The unit contains all of the global constants and variables, data structures, general purpose routines, and display routines. Once the

program starts executing, the code contained in the unit is always resident in memory.

The main program has also been divided into three segments. This division occured for two reasons: editor limitations and memory limitations. The solution provided an answer to both limitations. The three program segments do not have to be in memory at the same time because they perform independent functions. Each time their function is required, that segment of code is overlayed into memory and executed. The overlaying process takes no more than a few seconds and is accomplished at normal program break points. The source file for each segment also resides in a different text file; the files are compiled together into a single code file when the executive routine is compiled by using the compiler INCLUDE option. This allows a portion of a text file to exist as a separate text file, then at compilation time a single line in the main text file instructs the compiler to include the named file in this compilation. A change to the main program does not require the recompilation of the unit, but does require a current version of the unit to exist in the system library before a recompilation could be attempted. A change to any one of the segments would require recompiling the entire main program. The INCLUDE line must completely specify the file volume and name.

The program has borrowed from the example programs provided in the Apple Pascal Language Reference Manual. The specific routines are mentioned when discussed and in the program listing.

Notation. There has also been a change in the standard notation convention in the program. The reliability grid displayed by the program is dimensioned by an ordered pair (I,J). Any reference to I or the first dimension of the array IJPOS is to the column in the grid, while any reference to J or the second dimension of IJPOS is to the row in the grid. This convention is not normal (I normally indicates a row) but was noticed too late to change. The tedious precedure of reindexing was not considered worth the effort but may be accomplished by any enterprising programmer. The change could be made with a minimum of effort when the extension for the alternate data structure is attempted.

Diskette Detail. All the files discussed in this guide reside on the WORK: diskette. This diskette should be used as the source for all code; when changes are made, the files should be updated to reflect the changes. The WORKB: diskette is the WORK: diskette backup, it contains a copy of all of the files on the WORK: diskette. Once an updated version of the program is executing correctly and all the updated files are on the WORK: diskette, the WORKB: diskette should then be updated. In this fashion, only the

most recent change can be lost if the boot dikette is lost as the WORK: diskette is constantly being updated. If the WORK: diskette is damaged or lost, or the programmer has made an untraceable error, the WORKB: diskette can serve as backup, thus losing only the most recent test version of the program. To preclude the ultimate disaster, the WORKB: diskette should be stored separately.

Required Familiarity. This guide is written for a user who is familiar with the Apple implementation of the UCSD Pascal Operating System. The Apple Pascal Operating Reference Manual and the Apple Pascal Language Reference Manuals are necessary for complete understanding of the concepts and procedures used and discussed in this guide. The beginning user may also find Pascal Programming for the Apple by Lewis (Ref 10) to be very useful. The advanced user and the Pascal programmer will find PASCAL User Manual and Report by Jensen and Wirth (Ref 7) to be a complete reference.

## External Section

The external section will detail the process the user must go through to change and recompile the main program, and the slightly different process used to change, recompile, and relink the Basics unit. This discussion will assume that the user has a dual drive system.

### Main Program

The division of the main program into three text files makes editing and compiling unusual. The peculiarities of each will be discussed separately.

Editing. The editing process is the normal for each individual text file. Each text file has its own name, INITSEG and ANALSEG; the edit segment is contained in the main program file SYSREL. Each file can be edited and saved to any disk file in the normal fashion. The major differences occur during compiling.

Compiling. The compiling process is different because of the INCLUDE option. To use the include option, the user must completely specify the file name to be included. A copy of the INITSEG and ANALSEG files should be transferred to the APPLE2: diskette with the compiler because the INCLUDE option specifies the volume for these files as APPLE2:. This makes recompilation very easy. The APPLE2: diskette with the compiler and the include files INITSEG.TEXT and ANALSEG.TEXT should be placed in drive 2.

The APPLE1: diskette should be in drive 1 with the SYSREL.TEXT file as that file name or the SYSTEM.WRK.TEXT file along with the SYSTEM.LIBRARY file that contains the Basics unit. The work file or text file is then compiled normally. The compiler will go to the APPLE2: diskette and include the two separate segment text files when instructed to by the INCLUDE option in the main program text file. The names or the locations of the separate segment text files are not critical, but the specification of the file names on the include card must be exactly right: VOLUME:NAME.TEXT or the compilation will not work. The INCLUDE option is discussed in Chapter 4 on pages 63 and 64 in the Apple Pascal Language Reference Manual. The current version of the Basics unit must be linked into the system library or the compilation will not work. The RUN option may be used to compile and run the main program.

Basics Unit.

The process for the Basics unit is completely normal except for using the librarian utility provided on the APPLE3: diskette.

Editing. The editing process is normal as the entire text file will fit into the editor. The segment numbers chosen for this unit do not conflict with any other unit; care must be exercised if the segment numbers are changed to insure that there is no conflict with system provided segments. The Apple Pascal Language Reference Manual

discusses this point in detail in Chapter 5 on pages 76 and 77.

Compiling. The compilation process is also normal, except that the RUN command should not be used because the code file can not be executed until called by the main program. The compilation of a unit requires that the compilier SWAPPING option be turned on. This allows enough room in the computer for declarations but also slows the compilation process down. This point is discussed in Chapter 4 on page 68 in the Apple Pascal Language Reference Manual.

Linking. The linking process is simple to execute but possibly hard to understand. This guide will only present the procedure, an adequate explanation exists in the Apple Reference Manuals and Addendum. The procedure is easiest with the updated BASICS.CODE file on the WORK: diskette. This Diskette also contains the files OLD.LIBRARY and NEW.LIBRARY. These three files are accessed by the librarian utility to link the unit into a new library file. The user must now EXECUTE APPLE3:LIBRARY.CODE. The program will ask for output code file, reinsert the WORK: diskette and respond WORK:NEW.LIBRARY. The program will then ask for a link code file, respond with WORK:OLD.LIBRARY. Follow prompts and use the = command to link all source slots of the old library into the same destination slots of the new library. Now select a new link code file using the

N command and respond WORK:BASICS.CODE. Link slot 1 into slot 7 and slot 2 into slot 8. Now quit the librarian program using the Q command and the <RETURN> command. The NEW.LIBRARY file now contains an updated and linked version of the Basics unit. The SYSTEM.LIBRARY file on the APPLE1: diskette must now be replaced by the WORK:NEW.LIBRARY file. Once this has been accomplished the main program can be compiled or executed. The details of the librarian utility operation can be found in the Apple Pascal Operating System Manual in Chapter 8 on page 187.

## Internal Section: Basics Unit

This section will discuss the internal contents of the Basics unit. The topics will include unit structure, constants, types, variables, and data structures along with a discussion of each routine.

Structure. The structure of an intrinsic unit is very regimented. A unit has four parts: a heading, an interface part, an implementation part, and an initialization block. The heading of the Basics unit turns the compiler SWAPPING option on (required for all units), names the unit, declares the unit as an intrinsic unit, and specifies the segment numbers associated with the unit. The interface part declares the units this unit uses, the global constants, the global types, and the global variables the main program will use to communicate with all other segments and the unit. The interface part also declares all of the routines that are contained in the unit. The implementation part declares any local structures to the unit; the Basics unit has none. The implementation part also contains the body of each of the procedures or functions contained in the unit. The last part of a unit is an initialization block; this block is empty for the Basics unit. A detailed discussion of units is provided in Chapter 5 on pages 75 through 81 in the Apple Pascal Language Reference Manual.

Constants. The Basics unit declares seven constants that serve to size the entire program. The MAXX and MAXY constants are for the maximum X and Y positions allowed on the graphics screen. These values should not be changed as they are presently set to allow the maximum usable area on the graphics screen.

The NREC, NSER, and NPAR constants control the number of nodes allowed in the grid, the maximum number of components in a series structure, and the maximum number of components in a parallel structure. These constants may be adjusted by the user to suit the need. NREC is limited by the amount of memory available or the product of MAXI and MAXJ; memory currently allows for approximately 200 nodes while the grid could accept up to 400 nodes. NSER is limited to a single digit number. The single digit limitation is derived from the method of inputting the command parameter. This could be changed by adjusting the input format of the commands for parallel and series. NPAR is also limited to a single digit number but is used to determine how many parallel links each node must have, thus impacting the amount of memory each node must have. The current value of NPAR is 6, thus 6 words of memory are used for the parallel links of each node, as each link uses a word of memory.

The MAXI and MAXJ constants determine the column and row size of the reliability grid. These numbers may also be adjusted to suit the users need. A single restriction is that the MAXI value be 1 greater than the column size of the grid to allow for an invisible terminal node. Any reference to grid size excludes the extra column that the program actually needs. The minimum values should allow for a 10 by 10 grid; the maximum size the program can display at any one time. The current values of 20 by 20 allow for a demonstration of all of the program capablities.

Types. Pascal allows the user to declare nonstandard types to suit the programmers need and enhance self documentation. The Basics unit declares four new types: NODEPTR, NODE, CRTCOM, and CHARSET. The NODEPTR type defines variables that are used to point to variables of type NODE. The type NODE is a record data structure that contains the information pertaining to each component. The type CRTCOM declares the names of the variables that will be used as console commands. The type CHARSET declares variables that are of type SET OF CHARacter.

Variables. Pascal forces the programmer to declare all the variables the program uses before any executable statements of that block are compiled. Each block may have its own variables, but they are local (in force) only in that block or a subordinate block. Eight different types

of global variables are declared for the program.

The INTEGER type includes ICUR, JCUR, IMAX, JMAX, and NODES. ICUR and JCUR correspond to the column and row of the lower left grid postion in the current display. These values are changed as user moves though the grid, and then are used to provide the starting position of the display. IMAX keeps track of the rightmost column in the grid, while JMAX keeps track of the highest row in the grid. NODES keeps a running count of the number of nodes currently on the grid.

The SCREENCOLOR type includes only COLOR. This variable is used to indicate what color to draw the lines connectin he components on the display. It is currently set to WH.

The CHARacter type includes ANS, LAB1, and LAB2. ANS is a single character variable that is used to pick up keyboard inputs and transmit them to the program. LAB1 and LAB2 are also single character variables and are used to provide the letters the autolabelling procedure uses for component labels.

The BOOLEAN type includes AUTO and DATA. AUTO is a true or false variable that is used to indicate the status of the autolabelling option. DATA is used to indicate the status of the reliability grid: true indicates the grid contains data or the data was loaded from disk and false indicates that the grid is empty.

The REAL type only includes DREL. DREL is used to contain the default reliability selected by the user.

Three arrays of different types are also declared. CRTINFO is an ARRAY OF CHARacters indexed by the type CRTCOM that contains the console commands. PREFIXED is an ARRAY OF BOOLEAN indexed by the type CRTCOM that determines if the console command selected is prefixed by an escape character. IJPOS is an ARRAY OF NODEPTR that will be discussed as a data structure.

Data Structures. One type and one array serve as the programs only data structures. The type NODE represents each component. Each node is dynamically allocated as a new component is defined. This method uses a minimum amount of memory for a given size grid. The record structure used provides pointers for a forward and backward link, FLINK and BLINK, and NPAR parallel links in array PLINK. I, J are the integer grid column and row location of the component. ID1 and ID2 are the two character variables used for the component label. REL is a real number for the component reliability. Each link needs a single word of storage to represent any address in memory. I and J need only half a word to represent a integer that can be no larger than MAXI. ID1 and ID2 need half a word to represent a REL needs 2 words to represent a real number. Thus, a node currently needs 12 words of memory. This scheme could be changed by replacing the pointers with

(I,J) grid locations to allow for storage to disk and not change the memory requirements of each node.

The array IJPOS serves as the other data structure. This array, as sized by MAXI and MAXJ, provides the grid of pointers that determine the location of each of the components. This structure could possibly be combined with the record structure to make the disk to memory transfer a simple task. This idea is left as a program enhancement, which could discard the power of the dynamically allocated scheme in favor of a static array structure losing a substantial amount of the surplus memory avaliable for other enhancements.

Routines

The Basics unit contains 9 general purpose routines, 6 display routines, and 2 cursor movement routines. The display and cursor movement routines are edit segment functions but are placed in the unit to balance the size of the main program segments. The purpose of each routine will be briefly discussed; study of the program listing of each routine will provide the details.

CRT. The procedure CRT is used to carry out the specified console command. This routine was borrowed from the DISKIO program, which can be found on the Apple Pascal system diskette APPLE3:.

BACKUP. The procedure BACKUP is used to back the character cursor up on either the text or display screen depending on the option selected. This fuction enables the back arrow key. This routine is a modified version of a code sequence in the DISKIO program.

GETANS. The function GETANS is used to obtain a single character input from the user. The routine will echo the input to the text or graphics screen depending on the option selected. The routine also knows the only possible answers and will cycle until a correct response is obtained. This routine is also a modified version of a routine in the DISKIO program.

GETSTR. The procedure GETSTR is similar to GETANS and is used to obtain string input from the user. This routine will also echo the input to the correct screen. The original of this routine is found in the DISKIO program.

GETREL. The function GETREL is used to change an input string representing a real number into a real number represented by a variable. The function manipulates real number input strings such as .7239 into the correct internal representation between the values of 1.0 and 0.0. All three of the GET routines are sophisticated and deserve close attention.

DISPLAYAT. The procedure DISPLAYAT is used to communicate with the user on the display screen. The routine can clear the message area before displaying

depending on the option selected and can affect only a limited portion of the message area if necessary.

INSTRUCT. The procedure INSTRUCT is used to enable the <ESC> key to display the quick reference command list on the text screen. The words always exist on the text screen, the routine just switches from graphics display screen to the text screen to show them and then back to the unchanged reliability display.

AUTOLAB. The procedure AUTOLAB is used to automatically label a components ID if the autolabelling option is on. The default is on, but can be changed during program execution from the change menu.

INITNODE. The final general purpose procedure, INITNODE is used to initialize each node as required. The links are appropriately set, as are the reliability and label. The I and J positions are also set according to input parameters.

XPOS. The function XPOS is the first of the six display routines and is used to determine the X coordinate of the display screen position from the value of I for the component. This routine is scaled to provide for 10 components horizontally. Each component needs 28 dots and there are 280 dots available horizontally. The position on the screen is based on the current value of ICUR.

YPOS. The function YPOS is used as XPOS to determine the Y coordinate. This routine is scaled to provide 10 components vertically and a message area at the bottom of the display. Each component needs 18 dots and there are 192 dots available leaving 12 dots for the message area, or enough for more than a line of text.

CHECKDIS. The function CHECKDIS is used to check if the cursor component has been shifted out of the displayed grid. If outside the display, ICUR and JCUR are adjusted to center to current component and the function is set true. The value of the function is used to determine if the display should be changed. This routine is used in conjunction with DISPLAY to constantly keep the cursor component on the displayed part of the grid.

DCOMPON. The procedure DCOMPON is used to display an individual component in the mode selected. This routine provides the white on black or black on white components for the display. It is called by the DISPLAY routine.

DISPLAY. The procedure DISPLAY is used to display the grid on the graphics screen. The routine displays the grid starting from position (1,1). THis insures that all the connecting lines will be displayed. The components do not appear on the screen until the X and Y positions determined from XPOS and YPOS routines are plottable locations for the DCOMPON routine. This routine has two local procedures. DEXTEND draws the line from the component being displayed

to that components forward link. DASCEND draws the line from the component being displayed to the highest component that is in parallel.

POSITION. The procedure POSITION is the last display routine and is used to display the current row and column of the cursor component in the message area.

UPDOWN. The procedure UPDOWN is used to move the cursor component up or down on the display grid depending on the sign of the parameter. If no component is found at that location the routine checks to the right and left to find a target. If still unsuccessful the routine will move one row vertically towards the original component and try again. This can continue until a target or the original component is found.

LEFTRIGHT. The procedure LEFTRIGHT is used in the same manner as the UPDOWN routine but using horizontal moves and checking above and below.

## Internal Section: Main Program

The main program is divided into four parts: the executive routine, the initialize segment, the edit segment, and the analyze segment. The executive controls the three segments; each segment is an independent procedure that has its own executive and procedures. This section will discuss the executive routine and then each segment. The segment discussion will detail that segment's executive routine and procedures.

### Executive

The executive routine is responsible for scheduling the original order of the segments and then controls the main menu. The first function is nothing more than calling the initialize segment and then calling the edit segment. The generate sequence is executed if the user does not have data from disk and the edit sequence is executed if the user loaded the reliability grid with data from disk. Completing this task the executive routine becomes the main menu, selecting the correct segment based on the user input. The use of three independent segment procedures, each with there own executive routine, has made the operation of the main program executive very simple: the selection of the next segment to be executed or program termination.

## Initialize Segment

The initialize segment functions in two ways: first, to prepare the program for operation, and second, to allow the user to clear the grid and begin a new problem. This segment contains six procedures.

Executive. The initialize executive routine serves four functions. First, the GETCRT procedure is called the first time the initialize segment is called. This procedure determines the console commands. The option to return to the main menu is defeated the first time. Second, the global variables are all preset to their defaults. Third, the defaults can be changed by the user by selecting the correct option. And fourth, the data grid is initialized or loaded from disk depending on user selection. Control is then transfered back to the main program.

GETCRT. The procedure GETCRT is used to determine the console commands. The routine accesses the Pascal SYSTEM.MISCINFO file for the characters used as console commands. This file varies depending on the terminal system connected to the Apple. This routine will read the file for any type terminal and insure that the program has the correct commands. The CRTINFO and PREFIXED arrays are set to the appropriate values for use by the CRT procedure of the Basics unit. This routine was borrowed from the DISKIO program.

PROMPTAT. The procedure PROMPTAT is used to position the text screen cursor at a particular line and then write a line of text. This routine clears the keyboard buffer to insure the user does not type ahead. This routine was borrowed from the DISKIO program.

INTRO. The procedure INTRO is used to display the general system information on the text screen and present to the user as the program begins.

INITDISPLAY. The procedure INITDISPLAY is used to initialize the data grid array IJPOS and preset the display screen for use by the program. The first and invisible last nodes are inserted in their proper places to start the reliability grid.

GETDATA. The procedure GETDATA is not yet implemented. This routine would be used to query the user for the disk file from which to preload the reliability grid. This stub is provided with all the linkage necessary to operate in the program if this routine is implemented.

INFO. The procedure INFO is the last routine of the initialize segment and is used to write the quick reference command list to the text screen. This list is displayed by the INSTRUCT procedure in the BASICS unit.

Edit Segment

The edit segment has two functions: provide for the generate and edit sequences. All of the data manipulation occurs in the edit segment. This segment is by far the

largest in the program and has 16 subordinate routines.

Executive. The edit executive routine must insure a generate sequence is initiated if the data grid is empty, otherwise only an edit sequence occurs. The generate sequence is of the same form as the edit sequence. The current grid is displayed with the menu and the user is able to select a command. The generate sequence is limited to the P)arallel, S)eries, L)abel and N)ext commands. Completion of a generate sequence causes the edit menu to be displayed with no change to the grid and the edit sequence to be initiated. The full command set is now available, R)emove, C)hange, and the cursor movement commands, along with the Q)uit command to return to the main menu. Both sequences use this structure: check the cursor component for a grid change and display the change, highlight the cursor component and update the position information, display the menu and wait for an appropriate command, reset the cursor component and execute the command, repeat until the Q)uit command is selected. Each routine that has its own menu also functions in this same basic manner.

LABLE. The procedure LABLE (sic) is used to input the label and reliability for every component. The routine interacts through the message area on the display screen and the display changes immediately. The routine will automatically label a component if the option is turned on.

The routine converts the input string to the appropriate internal representation. If only a <RETURN> is input then the current value will be retained.

PARALLEL. The procedure PARALLEL is one of the recursive pair of routines that generate the grid. This routine has its own executive and five subordinate procedures: ERRORPAR, ABOVEPAR, CHECKPAR, SHIFTPAR, and INSERTPAR. The executive and routines perform in the following fashion. First, CHECKPAR determines if the commanded number can be input. The user hears two beeps if this can not be done, the routine takes appropriate action to complete the command. If no space is left the LABLE procedure is invoked and the procedure exits to the calling procedure. Second, ABOVEPAR allocates the amount of space needed to complete the command. Third, CHECKPAR reduces the amount to fit the available space and exits as in CHECKPAR if no space is available. Fourth, SHIFTPAR actually moves the components in the grid to make room for the new parallel structure. Fifth, INSERTPAR is used to place the possibly reduced amount into the grid. Finally, the basic executive sequence with the menu of S)eries, L)ABEL, or N)ext is executed with each inserted component as the cursor component. The other half of the recursive pair, SERIES, may now be called.

SERIES. The procedure SERIES is the other half of the recursive pair and is used to insert series structures into the grid. The routine has an executive routine and three subbordinate procedures: CHECKSER, SHIFTSER, and INSERTSER. The executive and the procedures perform in the following manner. First, the executive determines if a conflict exists due to the commanded number. Second CHECKSER provides error checking as in the PARALLEL routine and exits to the calling procedure if an error can not be resolved. Third, SHIFTSER moves the components in the grid to provide the needed space. Fourth, INSERTSER puts the possibly reduced number of components into the grid. Finally, the basic executive sequence is executed with the P)arallel, L)abel or N)ext commands for each component as in the PARALLEL routine. The PARALLEL routine may now be called recursively. The generate sequence alternates in the above fashion between the PARALLEL and SERIES routines.

REMOVE. The procedure REMOVE is used to remove a component from the grid. The routine is not very sophisticated and as such the routine can only be used in a certain sequence. To remove a series component, that component must not be the first component of a series placed in a parallel structure or the component that a parallel structure terminates before. To remove a parallel component, that component must a single component in parallel with other single components and not the lowest

component in the structure.

DEFAULT. The procedure DEFAULT is used to change to default reliability. The input string is entered through the display screen communication area and parsed to the appropriate internal value and stored in the variable DREL. If <RETURN> is the only input, the value of DREL is not changed.

CHANGE. The procedure CHANGE serves as a submenu for the edit sequence to provide for the commands P)arallel, S)eries, L)abel, D)efault, and T)oggle, along with Q)uit to return to the edit sequence menu. This routine takes the form of a basic executive sequence. The T)oggle command is simply switching the boolean variable AUTO from true to false or the reverse, thus controlling the autolabelling option. This routine will automatically exit to the edit menu after all but the T)oggle and D)efault commands.

Analyze Segment

The analyze segment has three functions: calculate the reliability of the grid, print a representation of the grid to paper, and store the grid to disk. This segment has a minimal executive that echos the choice made from the main menu for one of the three functions. This structure eliminates the need for a separate analyze menu. The segment contains nine subordinate procedures.

ANALYZE. The procedure ANALYZE is used to calculate the reliability of the system on the grid. This routine has three subordinate routines that recursively call one another to calculate the reliability: ANALMULT, ANALSER, and ANALPAR. First, ANALMULT is called by the analyze executive sequence to start the process. This routine multiplies the next structure in series by the current reliability. If the next structure is a parallel structure, ANALPAR is called to provide the reliability of that structure. ANALPAR in turn calls ANALSER to calculate the reliability of each series structure that make up the parallel structure. ANALSER recursively calls ANALMULT to multiply the structures in series together. This sequence is repeated as many times as necessary to reach the right most component in the grid. The ANALPAR routine is the critical part of the process and is very sophisticated in the manner of knowing when to close a parallel structure. The resulting reliability is displayed in the message area. The user can then print the grid by pressing the <RETURN> key or any other key to return to the main menu.

PRINT. The procedure PRINT uses the routines OUT and PAROUT to print a representation of the reliability grid to a piece of paper. The result is (I,J) grid locations of the components, the component reliability, and the components they are linked to and from. The analyze routine can be called from this routine.

STORDATA. The procedure STORDATA is the counterpart of the GETDATA procedure and as such is not yet implemented. The linkage for execution has been incorporated into the program.

## Procedure Hierarchy Chart

**Main Program**

- INITSEG
  - GETCRT
  - PROMPTAT
  - INTRO
  - INITDISPLAY
  - GETDATA
  - INFO
- EDITSEG
  - LABLE
  - PARALLEL
    - ERRORPAR
    - ABOVEPAR
    - CHECKPAR
    - SHIFTPAR
    - INSERTPAR
  - SERIES
    - CHECKSER
    - SHIFTSER
    - INSERTSER
  - REMOVE
  - DEFAULT
  - CHANGE
- ANALSEG
  - ANALYZE
    - ANALMULT
    - ANALSER
    - ANALPAR
  - PRINT
    - OUT
    - PAROUT
  - STORDATA

**Basics Unit**

- CRT
- BACKUP
- GETANS
- GETSTR
- GETREL
- DISPLAYAT
- INSTRUCT
- AUTOLAB
- INITNODE
- XPOS
- YPOS
- DCOMPON
- DISPLAY
  - DEXTEND
  - DASCEND
- POSITION
- UPDOWN
- LEFTRIGHT

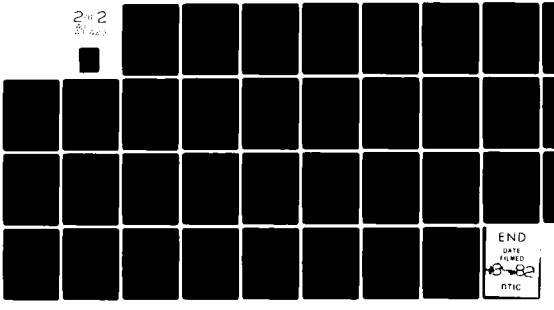
# Appendix A

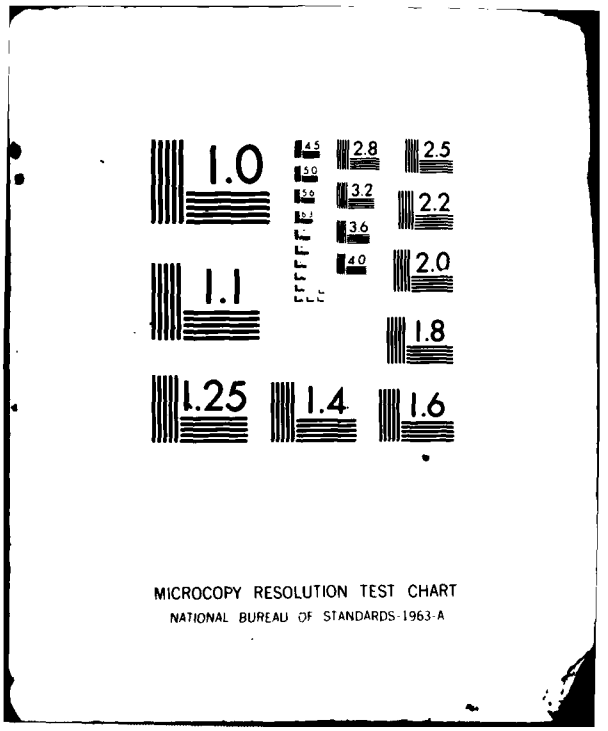
## Program listing

# Table of Contents

## Table of Contents

AD-A111 429 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
SYSTEM RELIABILITY: A MICROCOMPUTER SOLUTION TECHNIQUE.(U)
DEC 81 D R TUROS
UNCLASSIFIED AFIT/GOR/OS/81D-9 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
(*$S+*)  (* COMPILER SWAPPING OPTION MUST BE ON *)

UNIT BASICS; INTRINSIC CODE 25 DATA 26;

(*************************************************************)
(*                                                           *)
(*       THIS UNIT CONTAINS MANY OF THE SPECIAL PURPOSE      *)
(*  ROUTINES USED IN THE SYSTEM RELIABILITY PROGRAM.         *)
(*                                                           *)
(*       THE PROGRAM WAS WRITTEN BY CAPT DON TUROS           *)
(*  WHILE A STUDENT AT AFIT AS A THESIS EFFORT.  SOME        *)
(*  MATERIAL AND ALGORITHMS WERE BORROWED FROM EXAMPLES      *)
(*  PRESENTED IN THE APPLE ][ PASCAL OPERATING SYSTEM        *)
(*  MANUAL.  SPECIFIC MODULES ARE MENTIONED IN LINE.         *)
(*                                                           *)
(*************************************************************)

INTERFACE

USES   TURTLEGRAPHICS;(*  FOR ALL GRAPHICS *)

CONST  MAXX = 279;     (*  MAX X SCREEN POSITION  *)
       MAXY = 191;     (*  MAX Y SCREEN POSITION  *)
       NREC = 100;     (*  USER SET #   OF NODES  *)
       NSER = 8;       (*  USER SET # IN SERIES   *)
       NPAR = 6;       (*  USER SET # IN PARALLEL *)
       MAXI = 21;      (*  USER SET MAX COLUMNS   *)
                       (*  MUST BE # COL + 1      *)
       MAXJ = 20;      (*  USER SET MAX ROWS      *)

TYPE   NODEPTR = ^NODE;
       NODE = PACKED RECORD
                FLINK, BLINK : NODEPTR;
                PLINK : PACKED ARRAY[1..NPAR] OF NODEPTR;
                I,J   : 1..MAXI;
                ID1,ID2 : CHAR;
                REL   : REAL;
              END;
       CRTCOM = (EOS,EOL,UP,DN,RT,LT,LI);
       CHARSET = SET OF CHAR;

VAR    ICUR,JCUR,IMAX,JMAX,NODES : INTEGER;
       IJPOS : ARRAY[1..MAXI,1..MAXJ] OF NODEPTR;
       COLOR : SCREENCOLOR;
       ANS,LAB1,LAB2 : CHAR;
       AUTO,DATA : BOOLEAN;
       DREL : REAL;
       CRTINFO : PACKED ARRAY[CRTCOM] OF CHAR;
       PREFIXED : ARRAY[CRTCOM] OF BOOLEAN;
```

```
(*  GENERAL USAGE ROUTINES  *)

PROCEDURE CRT(C : CRTCOM);
PROCEDURE BACKUP(OPT : INTEGER);
FUNCTION  GETANS(OPT : INTEGER; OKSET :CHARSET) : CHAR;
PROCEDURE GETSTR(OPT :INTEGER; VAR S:STRING; MAX:INTEGER);
FUNCTION  GETREL(S : STRING) : REAL;
PROCEDURE DISPLAYAT(OPT : INTEGER; LOC:INTEGER; S:STRING);
PROCEDURE INSTRUCT;
PROCEDURE AUTOLAB(C : NODEPTR);
PROCEDURE INITNODE(I,J : INTEGER; VAR C : NODEPTR);

(*  DISPLAY ROUTINES        *)

FUNCTION  XPOS(I : INTEGER) : INTEGER;
FUNCTION  YPOS(J : INTEGER) : INTEGER;
FUNCTION  CHECKDIS(OPT : INTEGER; C : NODEPTR) : BOOLEAN;
PROCEDURE DCOMPON(C : NODEPTR; MODE : INTEGER);
PROCEDURE DISPLAY;
PROCEDURE POSITION(C : NODEPTR);

(*  CURSOR MOVEMENT ROUTINES  *)

PROCEDURE UPDOWN(J : INTEGER; VAR C : NODEPTR);
PROCEDURE LEFTRIGHT(I : INTEGER; VAR C : NODEPTR);
```

```
IMPLEMENTATION

PROCEDURE CRT;
(*  BORROWED FROM DISKIO                                  *)
(*  ACTIONS OCCURS AT THE CURRENT CURSOR POSITION  *)

BEGIN
  IF PREFIXED[C] THEN UNITWRITE(1,CRTINFO[LI],1,0,12);
  UNITWRITE(1,CRTINFO[C],1,0,12);
END;



PROCEDURE BACKUP;
(*  OPT = 1 -> BACKSPACE ON DISPLAY       *)
(*  ELSE    -> BACKSPACE ON TEXT SCREEN   *)

BEGIN
  IF OPT = 0 THEN
    BEGIN
      MOVETO(TURTLEX - 7,TURTLEY);
      WCHAR(' ');
      MOVETO(TURTLEX - 7,TURTLEY);
    END
  ELSE
    BEGIN
      CRT(LT);
      WRITE(' ');
      CRT(LT);
    END;
END;
```

```
FUNCTION GETANS;
(*  BORROWED FROM DISKIO                *)
(*  OPT = 1 -> ECHO TO TEXT SCREEN      *)
(*  OPT = 0 -> ECHO TO DISPLAY          *)
(*  ELSE    -> NO ECHO AT ALL           *)

VAR CH : CHAR;
    GOOD : BOOLEAN;

BEGIN
  REPEAT
    READ (KEYBOARD,CH);
    IF EOLN(KEYBOARD) THEN CH:=CHR(13);
    GOOD:=CH IN OKSET;
    IF NOT GOOD THEN WRITE(CHR(7))
      ELSE IF CH IN [' '..'Z'] THEN
        BEGIN
          IF OPT = 1 THEN WRITE(CH);
          IF OPT = 0 THEN WCHAR(CH);
        END;
  UNTIL GOOD;
  GETANS:=CH;
END;
```

```
PROCEDURE GETSTR;
(*  BORROWED FROM DISKIO                    *)
(*  OPT = 0 -> OPERATE ON DISPLAY           *)
(*  ELSE    -> OPERATE ON TEXT SCREEN       *)

VAR ANS : STRING[1];

BEGIN
  S:='';
  ANS:=' ';
  REPEAT
    IF LENGTH(S) = 0 THEN
      ANS[1]:=GETANS(OPT,[' '..'Z',CHR(13)])
    ELSE
      IF LENGTH(S) = MAX THEN
        ANS[1]:=GETANS(OPT,[CHR(8),CHR(13)])
      ELSE
        ANS[1]:=GETANS(OPT,[' '..'Z',CHR(8),CHR(13)]);
    IF ANS[1] IN [' '..'Z'] THEN
      S:=CONCAT(S,ANS)
    ELSE
      IF ANS[1] = CHR(8) THEN
        BEGIN
          BACKUP(OPT);
          DELETE(S,LENGTH(S),1);
        END;
  UNTIL ANS[1] = CHR(13);
END;
```

```
FUNCTION GETREL;

VAR  I,J : INTEGER;
     RL : REAL;

BEGIN
  RL:=0.0;
  IF S[1] = '1' THEN
    RL:=1.0
  ELSE
    BEGIN
      I:=POS('.',S) + 1;
      J:=10;
      WHILE I < (LENGTH(S) + 1) DO
        IF S[I] IN ['0'..'9'] THEN
          BEGIN
            RL:=RL + (ORD(S[I]) - 48) / J;
            J:=J * 10;
            I:=I + 1;
          END
        ELSE
          I:=LENGTH(S) + 1;
    END;
  GETREL:=RL;
END;
```

```
PROCEDURE DISPLAYAT;
(*  OPT = 1 - RESET GRAPHICS SCREEN MESSAGE AREA  *)

BEGIN
  UNITCLEAR(1);
  IF OPT = 1 THEN
    BEGIN
      VIEWPORT(LOC,MAXX,0,10);
      FILLSCREEN(BLACK);
      PENCOLOR(NONE);
      MOVETO(LOC,0);
      VIEWPORT(0,MAXX,0,MAXY);
    END;
  CHARTYPE(10);
  WSTRING(S);
END;



PROCEDURE INSTRUCT;

VAR  ANS : CHAR;

BEGIN
  TEXTMODE;
  ANS:=GETANS(2,[' '..'Z']);
  GRAFMODE;
END;
```

```
PROCEDURE AUTOLAB;

BEGIN
  IF AUTO AND (C^.ID1 = ' ') THEN
    BEGIN
      IF LAB2 = '9' THEN
        BEGIN
          LAB1:=SUCC(LAB1);
          IF LAB1 = 'Z' THEN LAB1:='A';
          LAB2:='0';
        END;
      LAB2:=SUCC(LAB2);
      C^.ID1:=LAB1;
      C^.ID2:=LAB2;
    END;
END;



PROCEDURE INITNODE;

VAR    K : INTEGER;

BEGIN
  NODES:=NODES + 1;
  NEW(C);
  C^.FLINK:=NIL;
  C^.BLINK:=NIL;
  FOR K:= 1 TO NPAR DO
    C^.PLINK[K]:=NIL;
  C^.ID1:=' ';
  C^.ID2:=' ';
  AUTOLAB(C);
  C^.I:=I;
  C^.J:=J;
  C^.REL:=DREL;
  IJPOS[I,J]:=C;
  IF NODES > NREC THEN
    BEGIN
      WRITE(CHR(7),CHR(7));
      DISPLAYAT(1,0,'# OF NODES EXCEED MAXIMUM');
      ANS:=GETANS(2,[' '..'Z']);
    END;
END;
```

```
FUNCTION XPOS;
(*  ALLOWS 10 HORIZONTAL COMPONENTS ON SCREEN  *)

BEGIN
  XPOS:=(I-ICUR)*28;
END;
```

```
FUNCTION YPOS;
(*  ALLOWS 10 VERTICAL COMPONENTS ON SCREEN  *)

BEGIN
  YPOS:=(J-JCUR)*18+18;
END;
```

```
FUNCTION CHECKDIS;
(*  OPT = 1  -  FIRST COMPONENT -> A NEW DISPLAY *)

VAR I,J : INTEGER;

BEGIN
  CHECKDIS:=FALSE;
  I:=C^.I;
  J:=C^.J;
  IF (( I > ICUR + 9 ) OR ( ICUR > I )) THEN
    BEGIN
      CHECKDIS:=TRUE;
      ICUR:=I - 5;
      IF ICUR <= 1 THEN
        ICUR:=1
      ELSE
        IF ICUR >= MAXI - 10 THEN
          ICUR:=MAXI - 10;
    END;
  IF (( J > JCUR + 9 ) OR ( JCUR > J )) THEN
    BEGIN
      CHECKDIS:=TRUE;
      JCUR:=J - 5;
      IF JCUR <= 1 THEN
        JCUR:=1
      ELSE
        IF JCUR >= MAXJ - 9 THEN
          JCUR:=MAXJ - 9;
    END;
  IF OPT = 1 THEN CHECKDIS:=TRUE;
END;
```

```
PROCEDURE DCOMPON;

VAR    X,Y,R : INTEGER;
       RL : STRING[2];

BEGIN
  PENCOLOR(NONE);
  X:=XPOS(C^.I);
  Y:=YPOS(C^.J);
  IF Y < 11 THEN EXIT(DCOMPON);
  MOVETO(X+3,Y);
  PENCOLOR(COLOR);
  MOVE(21);
  CHARTYPE(MODE);
  MOVETO(X+8,Y+1);
  WCHAR(C^.ID1);
  WCHAR(C^.ID2);
  MOVETO(X+8,Y-7);
  R:=ROUND(C^.REL*100);
  IF R = 100 THEN RL:='1 '
    ELSE
      BEGIN
        STR(R,RL);
        IF R < 10 THEN RL:=CONCAT('0',RL);
      END;
  WSTRING(RL);
END;
```

```
PROCEDURE DISPLAY;

VAR   C : NODEPTR;
      I,J : INTEGER;



PROCEDURE DEXTEND(C : NODEPTR);

VAR  NEW : NODEPTR;
     X,Y,XNEW,YNEW : INTEGER;

BEGIN
  X:=XPOS(C^.I);
  Y:=YPOS(C^.J);
  NEW:=C^.FLINK;
  XNEW:=XPOS(NEW^.I);
  YNEW:=YPOS(NEW^.J);
  PENCOLOR(NONE);
  MOVETO(X+24,Y);
  PENCOLOR(COLOR);
  MOVETO(XNEW-3,Y);
  IF YNEW = Y THEN
    MOVETO(XNEW+3,Y)
  ELSE
    MOVETO(XNEW-3,YNEW);
END;
```

```
PROCEDURE DASCEND(C : NODEPTR);

VAR  MAX : NODEPTR;
     J,X,Y,YMAX : INTEGER;
     DONE : BOOLEAN;

BEGIN
  MAX:=C;
  DONE:=FALSE;
  J:=1;
  REPEAT
    IF C^.PLINK[J] = NIL THEN DONE:=TRUE
      ELSE
        BEGIN
          IF MAX^.J < C^.PLINK[J]^.J THEN
            MAX:=C^.PLINK[J];
          J:=J + 1;
        END;
  UNTIL DONE OR (J = NPAR + 1);
  IF J <> 1 THEN
    BEGIN
      X:=XPOS(C^.I);
      Y:=YPOS(C^.J);
      YMAX:=YPOS(MAX^.J);
      PENCOLOR(NONE);
      MOVETO(X+3,Y);
      PENCOLOR(COLOR);
      MOVETO(X+3,YMAX);
    END;
END;
```

```
(* DISPLAY MAIN BODY *)
BEGIN
  INITTURTLE;
  VIEWPORT(0,MAXX,11,MAXY);
  FOR J:=1 TO JCUR+9 DO
    FOR I:=1 TO ICUR+9 DO
      BEGIN
        C:=IJPOS[I,J];
        IF (C <> NIL)  THEN
          IF C^.ID1 <> '*' THEN
            BEGIN
              DCOMPON(C,5);
              DEXTEND(C);
              DASCEND(C);
            END;
      END;
  VIEWPORT(0,MAXX,0,MAXY);
END;



PROCEDURE POSITION;

VAR IJ : STRING[2];

BEGIN
  PENCOLOR(NONE);
  CHARTYPE(10);
  MOVETO(0,0);
  IJ:='  ';
  STR(C^.J,IJ);
  IF C^.J < 10 THEN IJ:=CONCAT(' ',IJ);
  WSTRING(IJ);
  WCHAR('/');
  IJ:='  ';
  STR(C^.I,IJ);
  IF C^.I < 10 THEN IJ:=CONCAT(' ',IJ);
  WSTRING(IJ);
  WCHAR(' ');
END;
```

```
PROCEDURE UPDOWN;
(*  J CAN BE POSITIVE FOR UP AND NEGATIVE FOR DOWN  *)

VAR INEW,JNEW,INC : INTEGER;
    DONE : BOOLEAN;

BEGIN
  INEW:=C^.I;
  JNEW:=C^.J + J;
  IF JNEW > MAXJ THEN JNEW:=MAXJ;
  IF JNEW < 1 THEN JNEW:=1;
  (* POSITVE IF UP AND NEGATIVE IF DOWN *)
  IF J > 0  THEN
    INC:=1
  ELSE
    INC:=-1;
  DONE:=FALSE;
  REPEAT
    IF IJPOS[INEW,JNEW] <> NIL THEN
      DONE:=TRUE
    ELSE
      IF INEW > 1 THEN
        IF IJPOS[INEW-1,JNEW] <> NIL THEN
          BEGIN
            INEW:=INEW - 1;
            DONE:=TRUE;
          END
        ELSE
          IF INEW < MAXI - 1 THEN
            IF IJPOS[INEW+1,JNEW] <> NIL THEN
              BEGIN
                INEW:=INEW + 1;
                DONE:=TRUE;
              END;
    IF NOT DONE THEN JNEW:=JNEW - INC;
  UNTIL DONE;
  C:=IJPOS[INEW,JNEW];
END;
```

```
PROCEDURE LEFTRIGHT;
(*  LEFT IS NEGATIVE AND RIGHT IS POSITIVE  *)

VAR  INEW,JNEW,INC : INTEGER;
     DONE : BOOLEAN;

BEGIN
  INEW:=C^.I + I;
  JNEW:=C^.J;
  IF INEW > IMAX-1 THEN INEW:=IMAX - 1;
  IF INEW < 1 THEN INEW:=1;
  IF I > 0 THEN
    INC:=1
  ELSE
    INC:=-1;
  DONE:=FALSE;
  REPEAT
    IF IJPOS[INEW,JNEW] <> NIL THEN
      DONE:=TRUE
    ELSE
      IF JNEW < MAXJ THEN
        IF IJPOS[INEW,JNEW+1] <> NIL THEN
          BEGIN
            JNEW:=JNEW + 1;
            DONE:=TRUE;
          END
        ELSE
          IF JNEW > 1 THEN
            IF IJPOS[INEW,JNEW-1] <> NIL THEN
              BEGIN
                JNEW:=JNEW - 1;
                DONE:=TRUE;
              END;
    IF NOT DONE THEN INEW:=INEW - INC;
  UNTIL DONE;
  C:=IJPOS[INEW,JNEW];
END;




(* UNIT INITIALIZATION PART *)
BEGIN
  (*  NOTHING TO DO  *)
END.
```

```
(*$S+*)  (*  TURN COMPILER SWAPPING ON FOR SYMBOL TABLE *)

PROGRAM SYSREL;

(*************************************************************)
(*                                                           *)
(*       THIS PROGRAM PROVIDES THE USER WITH A TOOL TO       *)
(*  GENERATE, EDIT, ANALYZE, AND SAVE RELIABILITY            *)
(*  SYSTEMS.  THE SYSTEMS CAN CONTAIN SERIES AND             *)
(*  PARALLEL COMPONENTS, EACH IDENTIFIED WITH A USER         *)
(*  SUPPLIED ID TAG AND A CONSTANT RELIABILITY.  THE         *)
(*  PROGRAM WILL ALLOW A USER TO CONFIGURE A SYSTEM,         *)
(*  DETERMINE ITS RELIABILITY AND THEN CHANGE THE SYSTEM     *)
(*  A LA SENSITIVITY ANALYSIS TO DETERMINE METHODS  OF       *)
(*  ENHANCING RELIABILITY OR REDUCING THE NUMBER OF          *)
(*  COMPONENTS FOR THE SAME RELIABLITY.                      *)
(*                                                           *)
(*       THE PROGRAM WAS WRITTEN BY CAPT DON TUROS WHILE     *)
(*  A STUDENT AT AFIT AS A THESIS EFFORT.  SOME MATERIAL     *)
(*  AND ALGORITHMS WERE  BORROWED FROM EXAMPLES              *)
(*  PRESENTED IN THE APPLE ][ PASCAL OPERATING SYSTEM        *)
(*  MANUAL.  SPECIFIC MODULES ARE MENTIONED IN LINE.         *)
(*                                                           *)
(*************************************************************)

USES TURTLEGRAPHICS,BASICS;(* GRAPHICS AND USER ROUTINES *)

(*$I APPLE2:INITSEG.TEXT*) (* INCLUDE INITSEG HERE *)
```

```
SEGMENT PROCEDURE EDITSEG(OPT : BOOLEAN);
(*  OPT = TRUE -> THEN GRID HAS DATA IN IT  *)
(*  ELSE       -> NO DATA - RUN GENERATE    *)

VAR  C : NODEPTR;
     ANS,NUM : CHAR;
     REM,NI : INTEGER;



PROCEDURE LABLE(C : NODEPTR);

VAR    S : STRING;

BEGIN
  DISPLAYAT(1,42,'LABEL ->');
  IF AUTO THEN
    AUTOLAB(C)
  ELSE
    BEGIN
      DISPLAYAT(0,0,' ID = ');
      GETSTR(0,S,2);
      IF LENGTH(S) >= 1 THEN
        C^.ID1:=S[1];
      IF LENGTH(S) = 2 THEN
        C^.ID2:=S[2]
      ELSE
        C^.ID2:=' ';
    END;
  DISPLAYAT(0,0,' REL = ');
  (* MAX LENGTH IS 7 DIGITS AND DECIMAL POINT *)
  GETSTR(0,S,8);
  IF LENGTH(S) > 0 THEN
    C^.REL:=GETREL(S);
  DCOMPON(C,5);
END;

PROCEDURE SERIES(VAR NI : INTEGER; C : NODEPTR);FORWARD;
```

```
PROCEDURE PARALLEL(VAR NJ : INTEGER; C : NODEPTR);

VAR    I,J,NI,NP,IHI,ILOW,JHI,JLOW,JLAP : INTEGER;
       TODO : ARRAY[1..NPAR] OF NODEPTR;
       ABOVE,DONE,FOUND : BOOLEAN;
       ANS,NUM : CHAR;



PROCEDURE ERRORPAR;

BEGIN
  IF C^.J = MAXJ THEN
    BEGIN
      (* NO ROOM IN COLUMN *)
      WRITE(CHR(7),CHR(7));
      LABLE(C);
      EXIT(PARALLEL);
    END;
  IF C^.PLINK[NPAR] <> NIL THEN
    BEGIN
      (* NO PLINKS LEFT *)
      WRITE(CHR(7),CHR(7));
      LABLE(C);
      EXIT(PARALLEL);
    END
  ELSE
    BEGIN
      NP:=0;
      WHILE C^.PLINK[NP+1] <> NIL DO NP:=NP + 1;
      IF NJ - 1 > NPAR - NP THEN
        BEGIN
          (* TOO MANY - REDUCE FOR PLINKS *)
          WRITE(CHR(7),CHR(7));
          NJ:=NPAR - NP + 1;
        END;
    END;
END;
```

```
PROCEDURE ABOVEPAR;

VAR  NEW : NODEPTR;

BEGIN
  J:=C^.J;
  JLAP:=0;
  IHI:=C^.I;
  ILOW:=C^.I;
  JHI:=C^.J;
  JLOW:=C^.J;
  ABOVE:=FALSE;
  DONE:=FALSE;
  REPEAT
    I:=C^.I;
    J:=J + 1;
    FOUND:=FALSE;
    (* FIND FIRST COMPONENT ABOVE AND LEFT *)
    REPEAT
      IF IJPOS[I,J] <> NIL THEN
        FOUND:=TRUE
      ELSE
        I:=I - 1;
    UNTIL FOUND OR (I = 0);
```

```
      IF I = 0 THEN
        DONE:=TRUE
      ELSE
        BEGIN
          NEW:=IJPOS[I,J];
          IF NOT ABOVE THEN
            (* DETERMINE HOW MUCH ROOM AVAIL *)
            IF ((NEW^.I = C^.I) AND (NEW^.J = J)) OR
              (NEW^.FLINK^.I > C^.I) THEN
              BEGIN
                ABOVE:=TRUE;
                JLAP:=C^.J + NJ - J;
                JLOW:=J;
              END;
          (* FIND RIGHTMOST COMPONENT TO MOVE UP *)
          WHILE NEW^.FLINK^.J = J DO
            NEW:=NEW^.FLINK;
          IF NEW^.I > IHI THEN IHI:=NEW^.I;
          IF NEW^.FLINK^.I > C^.I THEN
            BEGIN
              JHI:=NEW^.J;
              (* FIND LEFTMOST TO MOVE UP *)
              NEW:=IJPOS[I,JJ];
              WHILE NEW^.BLINK^.J = J DO
                NEW:=NEW^.BLINK;
              IF NEW^.I < ILOW THEN ILOW:=NEW^.I;
            END;
        END;
    UNTIL DONE OR (J = MAXJ);
  END;
```

```
PROCEDURE CHECKPAR;

BEGIN
  IF DONE THEN
    (* NOT ENOUGH ROOM TO FIT ALL - REDUCE INPUT *)
    BEGIN
      IF JHI + JLAP > MAXJ THEN
        BEGIN
          WRITE(CHR(7),CHR(7));
          JLAP:=MAXJ - JHI;
          NJ:=JLAP + JLOW - C^.J;
        END;
      IF JHI + NJ - 1 > MAXJ THEN
        BEGIN
          WRITE(CHR(7),CHR(7));
          NJ:=MAXJ - JHI + 1;
        END;
    END
  ELSE
    BEGIN
      IF (JLOW - C^.J = 1) AND (JHI = MAXJ) THEN
        BEGIN
          (* NO ROOM PERIOD *)
          WRITE(CHR(7),CHR(7));
          LABLE(C);
          EXIT(PARALLEL);
        END;
      IF ABOVE THEN
        (* LIMITED ROOM - REDUCE *)
        BEGIN
          IF JHI + JLAP > MAXJ THEN
            BEGIN
              WRITE(CHR(7),CHR(7));
              JLAP:=MAXJ - JHI;
              NJ:=JLAP + JLOW - C^.J;
            END;
        END
      ELSE
        IF JLOW + NJ - 1 > MAXJ THEN
          BEGIN
            WRITE(CHR(7),CHR(7));
            NJ:=MAXJ - JLOW + 1;
          END;
    END;
END;
```

```
PROCEDURE SHIFTPAR;

BEGIN
  FOR J:=JHI DOWNTO JLOW DO
    FOR I:=ILOW TO IHI DO
      IF IJPOS[I,J] <> NIL THEN
        BEGIN
          IJPOS[I,J+JLAP]:=IJPOS[I,J];
          IJPOS[I,J]^.J:=J + JLAP;
          IJPOS[I,J]:=NIL;
        END;
END;
```

```
PROCEDURE INSERTPAR;

VAR  NEW : NODEPTR;

BEGIN
  NEW:=NIL;
  INITNODE(C^.I,C^.J+J-NP+1,NEW);
  C^.PLINK[J+1]:=NEW;
  NEW^.FLINK:=C^.FLINK;
  NEW^.BLINK:=C;
  TODO[J-NP+2]:=NEW;
  (* SET TOP COMPONENT FOR PRINTER *)
  IF NEW^.J > JMAX THEN JMAX:=NEW^.J;
END;
```

```
(* PARALLEL EXECUTIVE ROUTINE *)
BEGIN
  ERRORPAR;

  ABOVEPAR;

  CHECKPAR;

  IF JLAP > 0 THEN SHIFTPAR;

  TODO[1]:=C;
  FOR J:=NP TO NP + NJ - 2 DO
    INSERTPAR;

  FOR J:=1 TO NJ DO
    BEGIN
      IF CHECKDIS(J,TODO[J]) THEN DISPLAY;
      DCOMPON(TODO[J],10);
      POSITION(TODO[J]);
      DISPLAYAT(1,42,'PARALLEL -> S#,L,N -> ');
      REPEAT
        (*  PARALLEL MENU  *)
        NUM:='8';
        REPEAT
          ANS:=GETANS(0,['S','L','N',CHR(27)]);
          IF ANS = CHR(27) THEN INSTRUCT;
        UNTIL ANS <> CHR(27);
        IF ANS IN ['S'] THEN
          BEGIN
            NUM:=GETANS(0,['2'..CHR(NSER+48),CHR(8)]);
            IF NUM <> CHR(8) THEN
              NI:=ORD(NUM) - 48
            ELSE
              BACKUP(0);
          END;
      UNTIL NUM <> CHR(8);
      CASE ANS OF
        'S' : SERIES(NI,TODO[J]);
        'L' : LABLE(TODO[J]);
      END;
      DCOMPON(TODO[J],5);
    END;
END;
```

```
PROCEDURE SERIES;

VAR  I,J,ILAP,NJ : INTEGER;
     TODO : ARRAY[1..NSER] OF NODEPTR;
     NUM,ANS : CHAR;









PROCEDURE CHECKSER;

BEGIN
  IF ILAP + IMAX > MAXI - 1 THEN
    IF C^.FLINK^.I - C^.I = 1 THEN
      BEGIN
        (* NO ROOM AT ALL *)
        WRITE(CHR(7),CHR(7));
        LABLE(C);
        EXIT(SERIES);
      END
    ELSE
      BEGIN
        (* LIMITED ROOM - REDUCE *)
        WRITE(CHR(7),CHR(7));
        NI:=MAXI - 1 - IMAX + C^.FLINK^.I - C^.I - 1;
        ILAP:=C^.I + NI - C^.FLINK^.I;
      END;
END;
```

```
PROCEDURE SHIFTSER;

VAR  LAST : INTEGER;

BEGIN
  LAST:=C^.FLINK^.I;
  FOR I:=IMAX DOWNTO LAST DO
    FOR J:=1 TO MAXJ DO
      IF IJPOS[I,J] <> NIL THEN
        BEGIN
          IJPOS[I+ILAP,J]:=IJPOS[I,J];
          IJPOS[I,J]^.I:=I + ILAP;
          IJPOS[I,J]:=NIL;
        END;
  IMAX:=IMAX + ILAP;
END;
```

```
PROCEDURE INSERTSER;

VAR  NEW : NODEPTR;

BEGIN
  NEW:=NIL;
  INITNODE(C^.I+I-1,C^.J,NEW);
  NEW^.FLINK:=C^.FLINK;
  IF NEW^.FLINK^.J = NEW^.J THEN
    NEW^.FLINK^.BLINK:=NEW;
  NEW^.BLINK:=C;
  C^.FLINK:=NEW;
  TODO[I]:=NEW;
END;
```

```
(* SERIES EXECUTIVE ROUTINE *)
BEGIN
  ILAP:=C^.I + NI - C^.FLINK^.I;
  (* POSITIVE IF COMPONENT OVERLAP *)

  CHECKSER;

  IF ILAP > 0 THEN SHIFTSER;

  TODO[1]:=C;
  FOR I:= NI DOWNTO 2 DO
    INSERTSER;

  FOR I:= 1 TO NI DO
    BEGIN
      IF CHECKDIS(I,TODO[I]) THEN DISPLAY;
      DCOMPON(TODO[I],10);
      POSITION(TODO[I]);
      DISPLAYAT(1,42,'SERIES -> P#,L,N -> ');
      REPEAT
        (*  SERIES MENU  *)
        NUM:='6';
        REPEAT
          ANS:=GETANS(0,['P','L','N',CHR(27)]);
          IF ANS = CHR(27) THEN INSTRUCT;
        UNTIL ANS <> CHR(27);
        IF ANS IN ['P'] THEN
          BEGIN
            NUM:=GETANS(0,['2'..CHR(NPAR+48),CHR(8)]);
            IF NUM <> CHR(8) THEN
              NJ:=ORD(NUM) - 48
            ELSE
              BACKUP(0);
          END;
      UNTIL NUM <> CHR(8);
      CASE ANS OF
        'P' : PARALLEL(NJ,TODO[I]);
        'L' : LABLE(TODO[I]);
      END;
      DCOMPON(TODO[I],5);
    END;
END;
```

```
PROCEDURE REMOVE(VAR C : NODEPTR);

VAR  K : INTEGER;

BEGIN
  IF C^.BLINK^.FLINK = C THEN
    BEGIN
      IJPOS[C^.I,C^J]:=NIL;
      C^.BLINK^.FLINK:=C^.FLINK;
    END
  ELSE
    FOR K:= NPAR DOWNTO 1 DO
      IF C^.BLINK^.PLINK[K] = C THEN
        BEGIN
          IJPOS[C^.I,C^.J]:=NIL;
          C^.BLINK^.PLINK[K]:=NIL;
        END;
END;



PROCEDURE DEFAULT;

VAR  S : STRING;

BEGIN
  DISPLAYAT(1,0,'CHANGE DEFAULT RELIABILITY = ');
  GETSTR(0,S,8);
  IF LENGTH(S) > 0 THEN
    DREL:=GETREL(S);
END;
```

```
PROCEDURE CHANGE(C : NODEPTR);

VAR   ANS,NUM : CHAR;
      NI : INTEGER;

BEGIN
  REPEAT
    DCOMPON(C,10);
    POSITION(C);
    DISPLAYAT(1,42,'CHANGE -> P#,S#,L,D,T,Q -> ');
    REPEAT
      (*  CHANGE MENU  *)
      NUM:='6';
      (*  INSURE SERIES TO ALLOW PARALLEL  *)
      REPEAT
        IF (C^.FLINK^.J = C^.J)OR(C^.BLINK^.J = C^.J) THEN
          ANS:='P'
        ELSE
          ANS:='S';
        ANS:=GETANS(0,[ANS,'S','L','D','T','Q',CHR(27)]);
        IF ANS = CHR(27) THEN INSTRUCT;
      UNTIL ANS <> CHR(27);
      IF ANS IN ['P','S'] THEN
        BEGIN
          IF ANS = 'P' THEN
            NUM:=CHR(NPAR+48)
          ELSE
            NUM:=CHR(NSER+48);
          NUM:=GETANS(0,['2'..NUM,CHR(8)]);
          IF NUM <> CHR(8) THEN
            NI:=ORD(NUM) - 48
          ELSE
            BACKUP(0);
        END;
    UNTIL NUM <> CHR(8);
    CASE ANS OF
      'P' : PARALLEL(NI,C);
      'S' : SERIES(NI,C);
      'L' : LABLE(C);
      'D' : DEFAULT;
      'T' : AUTO:=NOT AUTO;
    END;
  UNTIL ANS IN ['Q','P','S','L'];
END;
```

```
(* EDIT EXECUTIVE ROUTINE *)
BEGIN
  C:=IJPOS[1,1];
  REM:=1;
  IF NOT OPT THEN
    BEGIN
      (*  GENERATE SEQUENCE  *)
      DISPLAY;
      DCOMPON(C,10);
      POSITION(C);
      DISPLAYAT(1,42,'GENERATE -> P#,S# -> ');
      REPEAT
        REPEAT
          ANS:=GETANS(0,['P','S',CHR(27)]);
          IF ANS = CHR(27) THEN INSTRUCT;
        UNTIL ANS <> CHR(27);
        IF ANS IN ['P'] THEN
          NUM:=CHR(NPAR+48)
        ELSE
          NUM:=CHR(NSER+48);
        NUM:=GETANS(0,['2'..NUM,CHR(8)]);
        IF NUM = CHR(8) THEN BACKUP(0);
      UNTIL NUM <> CHR(8);
      NI:=ORD(NUM) - 48;
      CASE ANS OF
        'P' : PARALLEL(NI,C);
        'S' : SERIES(NI,C);
      END;
    END;
```

```
(*  EDIT SEQUENCE  *)
REPEAT
  IF CHECKDIS(REM,C) THEN DISPLAY;
  REM:=0;
  DCOMPON(C,10);
  POSITION(C);
  DISPLAYAT(1,42,'EDIT -> R,C,I#,J#,K#,M#,H,Q -> ');
  REPEAT
    (*  EDIT MENU  *)
    NUM:='6';
    REPEAT
      ANS:=CHR(27);
      ANS:=GETANS(0,['R','C','I','J','K','M','H','Q',ANS]);
      IF ANS = CHR(27) THEN INSTRUCT;
    UNTIL ANS <> CHR(27);
    IF ANS IN ['I','J','K','M'] THEN
      BEGIN
        NUM:=GETANS(0,['1'..'9',CHR(8)]);
        IF NUM <> CHR(8) THEN
          NI:=ORD(NUM) - 48
        ELSE
          BACKUP(0);
      END;
  UNTIL NUM <> CHR(8);
  DCOMPON(C,5);
  CASE ANS OF
    'I' : UPDOWN(NI,C);
    'J' : LEFTRIGHT(-NI,C);
    'K' : LEFTRIGHT(NI,C);
    'M' : UPDOWN(-NI,C);
    'R' : REMOVE(C);
    'C' : CHANGE(C);
    'H' : C:=IJPOS[1,1];
  END;
  IF ANS = 'R' THEN
    BEGIN
      REM:=1;
      C:=C^.BLINK;
      IF C = NIL THEN
        C:=IJPOS[1,1];
    END;
UNTIL ANS IN ['Q'];
DATA:=TRUE;
END;
```

```
(*$I APPLE2:ANALSEG.TEXT*) (* INCLUDE ANAL SEGMENT HERE *)



(* MAIN PROGRAM EXECUTIVE *)
BEGIN
  INITSEG(TRUE);
  EDITSEG(DATA);
  REPEAT
    (*  MAIN MENU  *)
    IF DATA THEN
      BEGIN
        DISPLAYAT(1,0,'-> E)DIT,A)NAL,P)RINT,S)TORE,I)NIT,Q)UIT');
        REPEAT
          ANS:=CHR(27);
          ANS:=GETANS(0,['E','A','P','S','I','Q',ANS]);
          IF ANS = CHR(27) THEN INSTRUCT;
        UNTIL ANS <> CHR(27);
      END
    ELSE
      ANS:='E';
    CASE ANS OF
      'E' : EDITSEG(DATA);
      'A' : ANALSEG(1);
      'P' : ANALSEG(2);
      'S' : ANALSEG(3);
      'I' : INITSEG(FALSE);
    END;
  UNTIL ANS = 'Q';
GOTOXY(0,0);
CRT(EOS);
GOTOXY(0,15);
WRITELN('MEMORY AVAILIBLE - ',MEMAVAIL);
END.
```

```
(*********************************************************)
(*                                                       *)
(*       THIS SEGMENT PROVIDES FOR ALL THE INITIAL-      *)
(*   IZATION OR CLEARING OF THE DATA IN THE GRID.        *)
(*   THE GETCRT PROCEDURE PROVIDES TERMINAL INDE-        *)
(*   PENDENT CONSOLE CONTROL.  THE EXECUTIVE IS          *)
(*   RESPONSIBLE TO PROVIDE ALL THE DEFAULTS THE         *)
(*   PROGRAM NEEDS.  THE GETDATA STUB IS PROVIDE TO      *)
(*   ALLOW FOR INTERACTION WITH A DISK FILE IN THE       *)
(*   FUTURE.                                             *)
(*                                                       *)

SEGMENT PROCEDURE INITSEG(OPT : BOOLEAN);
(*  OPT = TRUE -> INITIAL TIME IN ROUTINE  *)
(*  ELSE       -> REINITIALIZE POSSIBILITIES  *)

VAR  S : STRING;
     ANS : CHAR;



PROCEDURE GETCRT;
(*  CODE USED FROM DISKIO PROGRAM  *)

VAR  I,BYTE : INTEGER;
     BUFFER : PACKED ARRAY[0..511] OF CHAR;
     F : FILE;

BEGIN
  RESET(F,'SYSTEM.MISCINFO');
  I:=BLOCKREAD(F,BUFFER,1);
  CLOSE(F);
  BYTE:=ORD(BUFFER[72]);
  CRTINFO[LI] :=BUFFER[62]; PREFIXED[LI] :=FALSE;
  CRTINFO[EOS]:=BUFFER[64]; PREFIXED[EOS]:=ODD(BYTE DIV 8);
  CRTINFO[EOL]:=BUFFER[65]; PREFIXED[EOL]:=ODD(BYTE DIV 4);
  CRTINFO[RT] :=BUFFER[66]; PREFIXED[RT] :=ODD(BYTE DIV 2);
  CRTINFO[UP] :=BUFFER[67]; PREFIXED[UP] :=ODD(BYTE);
  CRTINFO[LT] :=BUFFER[68]; PREFIXED[LT]:=ODD(BYTE DIV 32);
  CRTINFO[DN] :=CHR(10);    PREFIXED[DN] :=FALSE;
END;
```

```
PROCEDURE PROMPTAT(Y : INTEGER; S : STRING);
(*  BARROWED FROM DISKIO  *)

BEGIN
  UNITCLEAR(1);
  GOTOXY(0,Y);
  WRITE(S);
  CRT(EOL);
END;



PROCEDURE INTRO;

BEGIN
  GOTOXY(0,0);
  CRT(EOS);
  PROMPTAT(1,'RELIABILITY SYSTEM GENERATION PROGRAM');
  PROMPTAT(3,'SERIES AND PARALLEL STRUCTURES ONLY');
  PROMPTAT(4,'A MAX OF 6 COMPONENTS MAY BE IN PARALLEL');
  PROMPTAT(5,'THE SYSTEM MAY NOT EXCEED A 20 X 20 SIZE');
  PROMPTAT(6,'100 COMPONENTS MAXIMUM');
  PROMPTAT(7,'CURSOR COMPONENT IS WHITE ON BLACK');
  PROMPTAT(10,'DEFAULT FOR AUTOLABELLING IS ON');
  PROMPTAT(11,'DEFAULT FOR RELIABILITY IS 0.5');
  PROMPTAT(13,'<ESC> TO SHOW INSTRUCTIONS');
  PROMPTAT(14,'< <- > WORKS FOR A MULTI-KEYSTROKE INPUT');
  PROMPTAT(15,'<RETURN> GIVES CURRENT VALUE FOR LABEL');
  PROMPTAT(17,'SINGLE BEEP - INPUT ERROR');
  PROMPTAT(18,'DOUBLE BEEP - LIMIT ERROR');
  PROMPTAT(21,'ANY KEY TO CONTINUE');
  PROMPTAT(22,'<RETURN> TO CHANGE DEFAULTS');
  IF NOT OPT THEN
    PROMPTAT(23,'<ESC> TO RETURN TO MAIN MENU');
END;
```

```
PROCEDURE INITDISPLAY;

VAR   I,J : INTEGER;
      C : NODEPTR;

BEGIN
  C:=NIL;
  INITTURTLE;
  VIEWPORT(0,MAXX,0,MAXY);
  COLOR:=WHITE;
  DISPLAYAT(1,0,'PLEASE WAIT - INITIALIZING SYSTEM');
  FOR I:=1 TO MAXI DO
    FOR J:= 1 TO MAXJ DO
      IJPOS[I,J]:=NIL;
  ICUR:=1;
  JCUR:=1;
  IMAX:=2;
  JMAX:=1;
  INITNODE(2,1,C);
  C^.ID1:='*';
  C^.REL:=1.0;
  INITNODE(1,1,C^.BLINK);
  C^.BLINK^.FLINK:=C;
END;



PROCEDURE GETDATA;

VAR  ANS : CHAR;

BEGIN
  PROMPTAT(15,'ROUTINE NOT IMPLEMENTED');
  PROMPTAT(17,'ANY KEY TO CONTINUE');
  ANS:=GETANS(1,[' '..'Z']);
  DATA:=FALSE;
  INITDISPLAY;
END;
```

```
PROCEDURE INFO;

BEGIN
  GOTOXY(0,0);
  CRT(EOS);
  PROMPTAT(0,'E)DITING PROCESS COMMANDS');
  PROMPTAT(1,'GEN  P# -> # OF PARALLEL COMPONENTS');
  PROMPTAT(2,'     S# -> # OF SERIES COMPONENTS');
  PROMPTAT(3,'     L  -> LABEL CURRENT COMPONENT');
  PROMPTAT(4,'     N  -> MOVE TO NEXT COMPONENT');
  PROMPTAT(5,'EDIT R  -> REMOVE A COMPONENT');
  PROMPTAT(6,'     C  -> CHANGE A COMPONENT');
  PROMPTAT(7,'        P# -> AS ABOVE');
  PROMPTAT(8,'        S# -> AS ABOVE');
  PROMPTAT(9,'        L  -> AS ABOVE');
  PROMPTAT(10,'         D  -> DEFAULT CHANGE');
  PROMPTAT(11,'         T  -> TOGGLE AUTOLABELLING');
  PROMPTAT(12,'         Q  -> QUIT CHANGE');
  PROMPTAT(13,'     I#/M# -> GO UP/DOWN # LOCATIONS');
  PROMPTAT(14,'     J#/K# -> GO LEFT/RIGHT # LOCATIONS');
  PROMPTAT(15,'     H     -> GO TO LOWER LEFT LOCATION');
  PROMPTAT(16,'     Q  -> RETURN TO MAIN MENU');
  PROMPTAT(17,'A)NALYZE GRID RELIABILITY');
  PROMPTAT(18,'P)RINT GRID TO PAPER');
  PROMPTAT(19,'S)TORE GRID TO DISK');
  PROMPTAT(20,'I)INITIALIZING PROCESS');
  PROMPTAT(21,'Q)UIT PROGRAM');
  PROMPTAT(23,'ANY KEY TO RETURN');
END;
```

```
(* INITSEG EXECUTIVE ROUTINE *)
BEGIN
  IF OPT THEN GETCRT;
  INTRO;
  TEXTMODE;
  IF OPT THEN
    ANS:=CHR(13)
  ELSE
    ANS:=CHR(27);
  ANS:=GETANS(1,[' '..'Z',CHR(13),ANS]);
  IF ANS = CHR(27) THEN
    BEGIN
      GRAFMODE;
      INFO;
      EXIT(INITSEG);
    END;

  DREL:=0.5;
  LAB1:='A';
  LAB2:='/';
  NODES:=0;
  AUTO:=TRUE;
  DATA:=FALSE;

  IF ANS = CHR(13) THEN
    BEGIN
      GOTOXY(0,0);
      CRT(EOS);
      PROMPTAT(15,'AUTOLABELLING (Y OR N) = ');
      AUTO:=(GETANS(1,['Y','N']) IN ['Y']);
      PROMPTAT(15,'CHANGE THE DEFAULT RELIABILITY?');
      PROMPTAT(16,'ENTER <RETURN> FOR NO CHANGE OR');
      PROMPTAT(17,'ANY # FROM 1 TO .0000 :   REL = ');
      GETSTR(1,S,8);
      IF LENGTH(S) > 0 THEN DREL:=GETREL(S);
    END;
  GOTOXY(0,0);
  CRT(EOS);

  PROMPTAT(15,'GET DATA FROM DISK? (Y OR N) = ');
  DATA:=(GETANS(1,['Y','N']) IN ['Y']);
  IF DATA THEN
    GETDATA
  ELSE
    INITDISPLAY;
  INFO;
END;
```

```
(*************************************************************)
(*                                                           *)
(*       THIS SEGMENT PROVIDES THE AUTOMATED                 *)
(*  EQUATIONS TO DO THE RELIABILITY CALULATIONS.             *)
(*  THE STORDATA STUB PROVIDES FOR THE INTERFACE TO          *)
(*  DISK THAT WILL BE IMPLEMENTED IN THE FUTURE.             *)
(*  THIS ROUTINE ALSO PROVIDES THE PROCEDURE TO              *)
(*  PRINT THE SYSTEM TO HARD COPY.                           *)
(*                                                           *)

SEGMENT PROCEDURE ANALSEG(OPT : INTEGER);
(*  OPT = 1   ->  CALL ANALYZE ROUTINE     *)
(*  OPT = 2   ->  CALL PRINT ROUTINE       *)
(*  OPT = 3   ->  CALL STORDATA ROUTINE    *)

VAR  REL : REAL;

PROCEDURE PRINT(OPT : BOOLEAN); FORWARD;




PROCEDURE ANALYZE(OPT : BOOLEAN);
(*  OPT = TRUE  ->  CALL THE PRINT ROUTINE  *)
(*  ELSE        ->  CALLED FROM PRINT ROUTINE  *)

VAR  RL : STRING[4];
     I,J : INTEGER;

PROCEDURE ANALPAR(VAR I,J : INTEGER; VAR REL:REAL);FORWARD;
```

```
PROCEDURE ANALMULT(VAR I,J : INTEGER; VAR REL : REAL);

VAR   R : REAL;
      C : NODEPTR;

BEGIN
  C:=IJPOS[I,J];
  IF C^.PLINK[1] = NIL THEN
    BEGIN
      (* NEXT IS SINGLE COMPONENT *)
      REL:=REL * C^.REL;
      I:=C^.FLINK^.I;
      J:=C^.FLINK^.J;
    END
  ELSE
    BEGIN
      (* NEXT IS PARALLEL STRUCTURE *)
      (* ANALPAR UPATES I,J AND RELIABILITY *)
      R:=1;
      ANALPAR(I,J,R);
      REL:=REL * R;
    END;
END;



PROCEDURE ANALSER(VAR I,J : INTEGER; VAR REL : REAL);

VAR  ROW : INTEGER;

BEGIN
  ROW:=J;
  REL:=1;
  (* ANALMULT UPDATES I,J AND RELIABILITY *)
  REPEAT
    ANALMULT(I,J,REL);
  UNTIL J < ROW;
END;
```

```
PROCEDURE ANALPAR;

VAR  SERREL : ARRAY[1..NPAR] OF REAL;
     SEREND : ARRAY[1..NPAR] OF INTEGER;
     K,HI,NJ,ROW : INTEGER;
     R : REAL;
     C : NODEPTR;
     DONE,CLOSE : BOOLEAN;

BEGIN
  (* SERREL HAS SERIES RELIABILITY *)
  (* SEREND HAS LAST SERIES COMPONENT LOCATION *)
  HI:=I;
  C:=IJPOS[I,J];
  I:=C^.FLINK^.I;
  J:=C^.FLINK^.J;
  REL:=C^.REL;
  DONE:=FALSE;
  NJ:=1;
  REPEAT
    (* CHECK FOR PARALLEL STRUCTURES *)
    IF C^.PLINK[NJ] = NIL THEN
      DONE:=TRUE
    ELSE
      BEGIN
        SERREL[NJ]:=0.0;
        SEREND[NJ]:=HI;
        NJ:=NJ + 1;
      END;
  UNTIL DONE OR (NJ > NPAR);
  NJ:=NJ - 1;

  (* DETERMINE SERIES RELIABILITY *)
  FOR K:=1 TO NJ DO
    BEGIN
      ROW:=C^.PLINK[K]^.J;
      ANALSER(SEREND[K],ROW,SERREL[K]);
      IF SEREND[K] > HI THEN HI:=SEREND[K];
    END;
```

```
(* DETERMINE PARALLEL RELIABILITY *)
DONE:=FALSE;
REPEAT
  C:=IJPOS[I,J];
  CLOSE:=FALSE;
  FOR K:=1 TO NJ DO
    IF I = SEREND[K] THEN CLOSE:=TRUE;
  (*  CLOSE = TRUE -> CLOSE A PARALLEL STRUCTURE  *)
  IF CLOSE THEN
    BEGIN
      R:=1;
      (* DETERMINE PARALLEL BASE RELIABILITY *)
      FOR K:=1 TO NJ DO
        IF I = SEREND[K] THEN
          BEGIN
            R:=R * (1 - SERREL[K]);
            SEREND[K]:=0;
          END;
      (* PARALLEL RELIABLITY EQUATION *)
      REL:=1 - (1 - REL) * R;
      IF I = HI THEN DONE:=TRUE;
    END
  ELSE
    ANALMULT(I,J,REL);
UNTIL DONE;
END;
```

```
(* ANALYZE EXECUTIVE ROUTINE *)
BEGIN
  DISPLAYAT(1,0,'PLEASE WAIT - ANALYZING THE SYSTEM');
  I:=1;
  J:=1;
  REPEAT
    ANALMULT(I,J,REL);
  UNTIL I >= IMAX;
  DISPLAYAT(1,0,'RELIABILITY = ');
  (* 10000 TO GIVE 4 DIGITS *)
  I:=ROUND(REL * 10000);
  IF I = 10000 THEN
    WSTRING('1.0')
  ELSE
    BEGIN
      STR(I,RL);
      IF I < 1000 THEN RL:=CONCAT('0',RL);
      IF I < 100 THEN RL:=CONCAT('0',RL);
      IF I < 10 THEN RL:=CONCAT('0',RL);
      WCHAR('.');
      WSTRING(RL);
    END;
  IF OPT THEN
    WSTRING(' - <RET> FOR PRINT')
  ELSE
    WSTRING(' - ANY KEY');
  IF (GETANS(2,[' '..'Z',CHR(13)]) IN [CHR(13)]) AND OPT THEN
    PRINT(FALSE);
END;
```

```
PROCEDURE PRINT;
(*  OPT = TRUE -> CALL ANALYZE ROUTINE     *)
(*  ELSE       -> CALLED BY ANALYZE ROUTINE  *)

VAR  LEFT,RIGHT : INTEGER;
     ANS : CHAR;
     P : TEXT;
     S : STRING;



PROCEDURE OUT(TOP,BOT,LEFT,RIGHT : INTEGER);

VAR I,J,K : INTEGER;
    C : NODEPTR;
    B8 : STRING[8];



PROCEDURE PAROUT(OPT : INTEGER);
(*  OPT = ODD  -> FIRST LINK OF POSSIBLE PAIR  *)
(*  OPT = EVEN -> SECOND LINK OF PAIR          *)

BEGIN
  FOR J:=LEFT TO RIGHT DO
    BEGIN
      C:=IJPOS[J,I];
      IF C = NIL THEN
        WRITE(P,B8)
      ELSE
        BEGIN
          C:=C^.PLINK[OPT];
          IF C = NIL THEN
            WRITE(P,B8)
          ELSE
            IF ODD(OPT) THEN
              WRITE(P,' ^',C^.J:2,'    ')
            ELSE
              WRITE(P,'    /',C^.J:2,' ');
        END;
    END;
END;
```

```
(* PROCEDURE OUT MAIN BODY.*)
BEGIN
  B8:='        ';
  (* WRITE - NO CR OR LF *)
  (* WRITELN - CR AND LF *)
  FOR I:=TOP DOWNTO BOT DO
    BEGIN
      (* ROW/COL POSITION *)
      FOR J:=LEFT TO RIGHT DO
        BEGIN
          C:=IJPOS[J,I];
          IF C = NIL THEN
            WRITE(P,B8)
          ELSE
            WRITE(P,' *',C^.J:2,'/',C^.I:2,' ');
        END;
      WRITELN(P);
      (* RELIABILITY #.#### *)
      FOR J:=LEFT TO RIGHT DO
        BEGIN
          C:=IJPOS[J,I];
          IF C = NIL THEN
            WRITE(P,B8)
          ELSE
            WRITE(P,C^.REL:7:4,' ');
        END;
      WRITELN(P);
      (* ROW/COL OF FORWARD LINK *)
      FOR J:=LEFT TO RIGHT DO
        BEGIN
          C:=IJPOS[J,I];
          IF C = NIL THEN
            WRITE(P,B8)
          ELSE
            BEGIN
              C:=C^.FLINK;
              IF C = NIL THEN
                WRITE(P,B8)
              ELSE
                WRITE(P,' >',C^.J:2,'/',C^.I:2,' ');
            END;
        END;
      WRITELN(P);
```

```
        (* ROW/COL OF BACK LINK *)
        FOR J:=LEFT TO RIGHT DO
          BEGIN
            C:=IJPOS[J,I];
            IF C = NIL THEN
              WRITE(P,B8)
            ELSE
              BEGIN
                C:=C^.BLINK;
                IF C = NIL THEN
                  WRITE(P,B8)
                ELSE
                  WRITE(P,' <',C^.J:2,'/',C^.I:2,' ');
              END;
          END;
        WRITELN(P);
        (* ROW OF PARALLEL LINKS - SAME COLUMN *)
        K:=1;
        REPEAT
          IF K <= NPAR THEN
            PAROUT(K);
          (* CR BUT NO LF FOR EVEN PLINK *)
          (* CHR(13) ALWAYS HAS LF SO MUST USE *)
          (* CHR(141) IN PASCAL *)
          WRITE(P,CHR(141));
          IF K + 1 <= NPAR THEN
            PAROUT(K+1);
          WRITELN(P);
          K:=K + 2;
        UNTIL K > NPAR;
        (* BLANK LINE BETWEEN ROWS *)
        WRITELN(P);
      END;
END;
```

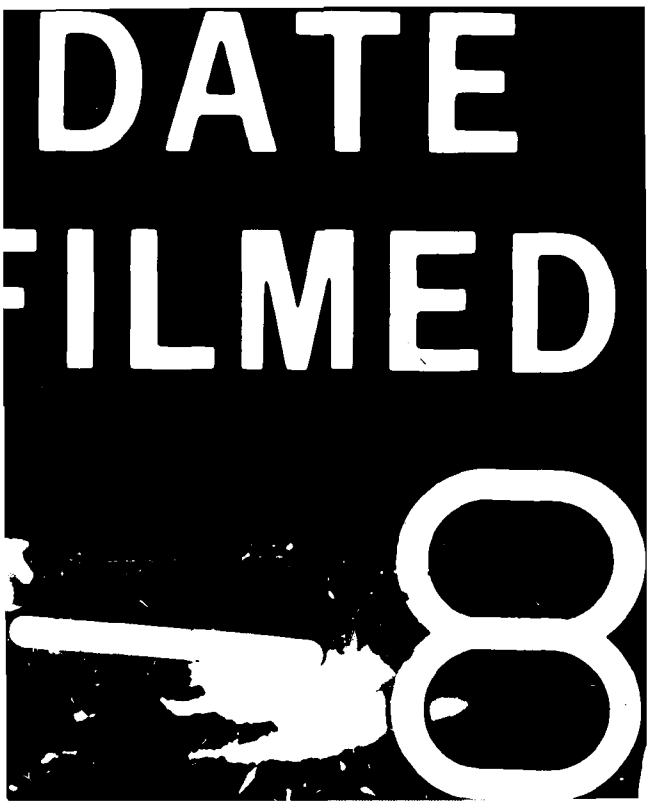
```
(* PRINT EXECUTIVE ROUTINE *)
BEGIN
  REWRITE(P,'PRINTER:');
  DISPLAYAT(1,0,'ENTER PRINT TITLE -> ');
  GETSTR(0,S,20);
  IF LENGTH(S) = 0 THEN
    S:=CONCAT(' TEST SYSTEM ');
  IF OPT THEN
    BEGIN
      DISPLAYAT(1,0,'ANALYZE SYSTEM (Y OR N) -> ');
      ANS:=GETANS(0,['Y','N',CHR(27)]);
      IF ANS IN ['Y'] THEN
        ANALYZE(FALSE)
      ELSE
        IF ANS = CHR(27) THEN
          EXIT(PRINT);
    END;
  DISPLAYAT(1,0,'PLEASE WAIT - PRINTING GRID');
  LEFT:=1;
  REPEAT
    IF IMAX - 1 <= LEFT + 9 THEN
      RIGHT:=IMAX - 1
    ELSE
      RIGHT:=LEFT + 9;
    WRITE(P,'     RELIABILITY SYSTEM:    ',S);
    WRITELN(P,'   RELIABILITY = ',REL:7:4);
    WRITELN(P);
    OUT(JMAX,1,LEFT,RIGHT);
    PAGE(P);
    LEFT:=LEFT + 10;
  UNTIL LEFT > IMAX - 1;
  CLOSE(P);
END;
```

```
PROCEDURE STORDATA;

VAR  ANS : CHAR;

BEGIN
  DISPLAYAT(1,0,'ROUTINE NOT IMPLEMENTED  -  ANY KEY');
  ANS:=GETANS(2,[' '..'Z']);
END;
```

```
(* ANALSEG EXECUTIVE ROUTINE *)
BEGIN
  REL:=1.0;
  CASE OPT OF
    1 : ANALYZE(TRUE);
    2 : PRINT(TRUE);
    3 : STORDATA;
  END;
END;
```

# Vita

Donald R Turos, Jr was born in Cleveland, Ohio and raised on the San Francisco peninsula. He attended the Air Force Academy and received a Bachelor of Science in Computer Science and his regular commission in the Air Force on 1 June 1977. He served at the Tactical Fighter Weapons Center Range Group, Nellis AFB, Nevada until entering the Air Force Institute of Technology in June of 1980. During his tour at Range Group, he was responsible for data reduction software for the Nellis Range Complex in support of any range activity and, in particular the Redflag exercises.


DATE
FILMED
8